UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Cluster in a box: z Systems, COBOL y DB2

Pablo Pérez de Madariaga
Tutor: Estrella Pulido Cañabate

Enero 2016

Cluster in a box: z Systems, COBOL y DB2

AUTOR: Pablo Pérez de Madariaga
TUTOR: Estrella Pulido Cañabate

Dpto. Ingeniería informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Enero de 2016

Resumen (castellano)

El objetivo de este trabajo de fin de grado es el estudio de un entorno no muy conocido en el ámbito de la informática, el entorno mainframe. Haremos una presentación de todo el entorno: software y hardware. También ilustraremos cómo desarrollar aplicaciones en lenguaje COBOL y en el sistema de base de datos DB2 y realizaremos mediciones para evaluar su comportamiento en este entorno.

Primero describimos la parte hardware de los entornos mainframe, es decir, las máquinas que dan servicio a estos entornos, mostrando las características de las mismas e indicando los motivos que las hacen ideales para grandes negocios. En esta parte describimos características tales como el RAS (reliability, availavility y serviceability) que son cruciales a la hora de que las grandes empresas opten por este sistema para albergar su negocio.

Introduciremos el sistema operativo del mainframe, haremos una descripción detallada con un enfoque cada vez más minucioso del sistema, describiendo componentes como el WLM o Workload Management, un componente vital a la hora de optimizar y gestionar todos los recursos del sistema y así cumplir con eficacia los objetivos de negocio.

A medida que vamos describiendo el entorno vamos presentando las tecnologías que se explotan en él, como el lenguaje COBOL, JCL o el DB2 entre otros. Describiremos cómo realizar aplicaciones y preparar nuestro entorno para desarrollar utilizando estas tecnologías.

En nuestro estudio, evaluaremos los lenguajes COBOL y C, los compararemos desde diferentes puntos de vista y haremos programas para medirlos y llegar a una conclusión sobre sus ventajas y desventajas.

Una vez que terminamos la comparación de los lenguajes procederemos a ilustrar el DB2 del mainframe. Describiremos a este gestor de bases de datos y lo probaremos mediante programas COBOL, evaluando sus características para desarrollar aplicativos. Para realizar esta evaluación, construimos programas en COBOL que prueban diferentes casos y operativas del lenguaje SQL. Mediremos tiempos de consultas en un sistema de tablas de diferentes tamaños, algunos realmente grandes, para posteriormente, poder dar una evaluación del gestor de bases de datos del mainframe.

Finalmente se concluye el estudio con una valoración de las tecnologías usadas con propuestas para trabajos futuros, centrándonos en la parte online del mainframe, ya que durante nuestro estudio hemos probado la parte batch o de procesamiento por lotes.

Abstract (English)

The target of this bachelor thesis is the study of a not very known environment in computer science, the mainframe environment. We will first present the environment: software and hardware. Then we will show how to develop applications in both languages, COBOL and DB2, all this accompanied by measurements to evaluate their behavior in this environment.

First of all we focus on describing the hardware part of the mainframe environments, it means, we focus on the machines that give service to these environments, showing their features and stating the reasons why they are ideal for huge business. In this part we describe features such as the “RAS” (reliability, availability and serviceability) and how they are essential for big companies at the time of choosing this system to host their business.

We will introduce the mainframe operating system and we will give a full description with a detailed approach of the system, describing components like WLM or Workload Management, a critical component at the time of optimizing and managing all system resources to achieve the business goals.

As we describe the environment, we introduce several technologies used in it. Some of them are COBOL, JCL or DB2. We then represent how to create applications and how to get our environment ready for developing in these technologies.

In our study, we will evaluate COBOL and C languages, comparing them from different points of view and we will create programs for these comparatives getting to a final conclusion about their advantages and disadvantages.

Once we have finished the comparative of both languages, we will study DB2 describing this Database manager and testing it with COBOL programs, evaluating its characteristics. To accomplish this evaluation, we build up programs in COBOL that test different cases and procedures of the SQL language. We will quantify the time a system takes to access different tables so that we can give a real evaluation of the database manager of the mainframe.

The study finishes with an overall assessment of the technologies used. In addition, we add suggestions for future works, focusing on the online part of the mainframe due to in our study we have used the batch part or the batch processing.

Palabras clave (castellano)

Mainframe, COBOL, z Systems, DB2, disponibilidad, seguridad, fiabilidad, escalabilidad, rendimiento.

Keywords (inglés)

Mainframe, COBOL, z Systems, DB2, availability, safety, reliability, scalability, optimize.

Agradecimientos

Me gustaría empezar agradeciendo a todas las personas que en todos estos años han estado conmigo durante la carrera, en especial a esas pequeñas “criaturas” que sin ellas, estoy seguro que todo esto no se hubiera llevado a cabo, o no de la misma manera. Sois estupendos chicos muchas gracias a todos por estos años de carrera a vuestro lado.

En segundo lugar me gustaría agradecer a todos mis amigos y sobre todo a aquellas personas que en la realización de este trabajo se han volcado en animarme y ayudarme cuando lo necesitaba, han sido un apoyo constante, sin estas personas esto no hubiera sido posible, mil gracias a ellos.

En tercer lugar me gustaría agradecer este trabajo a todos los profesionales, profesores y tutores que se han implicado conmigo durante la carrera, han demostrado tener paciencia y dedicación no solo conmigo, con todos sus alumnos y eso en sus clases se nota la voluntad de enseñar y la vocación con la que viven la enseñanza, muchas gracias a ellos.

Por último también a toda la gente de IBM que me ha ayudado y enseñado, a los profesionales de Everis e Isban, que me han formado, prestado su ayuda y aguantado con paciencia todas mis preguntas.

Gracias a todos.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

En nuestra vida cotidiana usamos muchos aplicativos que tienen que ver con un entorno Mainframe, al sacar dinero, al consultar nuestra cartilla del banco…. Es nuestro objetivo indagar en este tema, en lo que hay detrás de un simple cajero automático, en el lugar donde las grandes empresas y los multimillonarios bancos alojan toda su lógica de negocio.

¿Qué entorno usarán? Para albergar información tan susceptible como el dinero en si mismo necesitarán un entorno diferente al resto, ¿o no?, algo que sea más seguro de lo normal, que no se quede colgado a la hora de la verdad, que sea capaz de hacer transacciones de millones de euros sin peligro alguno. ¿Cómo lo hacen? ¿Qué lenguaje de programación usan sus programas? Si nos ponemos a pensar nos surgen infinidad de dudas al respecto.

1.2 Objetivos

En este trabajo de fin de grado vamos a intentar dar una introducción a este mundo desconocido del mainframe, el mundo del "ordenador central", término acuñado hace décadas para referirse a ordenadores de tamaño y características muy superiores a los ordenadores personales que se comercializaban entonces, a lo que hay detrás de una simple terminal de cajero automático. Y después de introducirnos un poco en este mundo del mainframe haremos una comparativa de dos lenguajes de programación: C y COBOL.

Esta comparativa de lenguajes no la hemos elegido al azar, el COBOL es un lenguaje con fama de "viejo" y "obsoleto". Sin embargo, la fama que tiene está totalmente equivocada, COBOL sigue activo y muy vivo, se siguen desarrollando multitud de aplicaciones y programas en COBOL actualmente. Nuestro siguiente candidato es el lenguaje C. Lo elegimos por ser compilado, como COBOL, para poder hacer una comparativa equilibrada.

La comparativa la haremos compilando y ejecutando en un entorno mainframe de IBM real, el "Marist College", situado en Estados Unidos. Veremos qué lenguaje es más rápido a la hora de procesar ficheros, que es uno de los casos que más se usan en el sector de negocios grandes que manejan y tratan grandes volúmenes de datos; como abrir un fichero, procesar los datos y cerrarlo, una tarea sencilla que se puede convertir perfectamente en un incordio si los ficheros son del orden de cientos de miles de registros, una casuística real que se hace habitualmente en el sector bancario.

Para finalizar nuestro estudio no podemos dejar de lado las bases de datos. En nuestro entorno mainframe daremos una visión global de la base de datos que aloja nuestra máquina de IBM, el DB2. Veremos cómo acceder a él y ejecutaremos alguna consulta en su base de datos.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Bloque I**
  - **Capítulo 1: Mainframes**
  - **Capítulo 2: El sistema operativo z/OS**
  - **Capítulo 3: La programación en mainframes**

- **Bloque II**
  - **Capítulo 1: Creación de programas en el mainframe**
  - **Capítulo 2: Comparativa de lenguajes: COBOL & C**

- **Bloque III**
  - **Capítulo 1: JCL de compilación y lanzado de programas**
  - **Capítulo 2: Programas de prueba**

- **Bloque IV**
  - **Capítulo 1: Programas sin DB2**
  - **Capítulo 2: Programas con DB2**

- **Bloque V**
  - **Capítulo 1: Conclusiones**
  - **Capítulo 2: Trabajo futuro**

2 Estado del arte

2.1 Mainframes

2.1.1 ¿Qué es un mainframe?

En el diccionario de IBM, "mainframe" se describe de la siguiente manera:

"*A large computer, in particular one to which other computers can be connected so that they can share facilities. The mainframe provides (for example, a System/370 computing system to which personal computers are attached so that they can upload and download programs and data). The term usually refers to hardware only, namely, main storage, execution circuitry and peripheral units*"

UAM-IBM grandes sistemas corporativos mainframe.

Lo que en Castellano viene a decir: "Un mainframe es un ordenador grande, al cual otros ordenadores pueden estar conectados de tal manera que pueden compartir características y funcionalidades de muchas maneras. Por ejemplo, un mainframe puede proporcionar a los ordenadores a los que está conectado la posibilidad de cargar/descargar programas y datos de él. El término suele referirse a hardware únicamente (almacenamiento principal, circuitería de ejecución y periféricos)".

Pero esta definición es muy "teórica". Podríamos perfectamente definir un mainframe como el lugar donde las empresas alojan sus bases de datos comerciales, servidores transaccionales y aplicaciones que requieren un mayor nivel de seguridad del que pudieran proporcionarles otras máquinas más pequeñas. Sobre seguridad del mainframe y su S.O puede consultar [16].

A estas características de alta seguridad se le suman, entre otras, una enorme capacidad de procesamiento de trabajos y transacciones, dando como resultado las siguientes estadísticas sobre el uso de mainframes.

Las 1000 compañías más importantes de todo el mundo utilizan para su negocio el entorno mainframe:

- **97** de los **100** bancos más importantes de todo el mundo usan mainframe
- **23** de los **25** minoristas más importantes del mundo usan mainframe
- **10** de las **10** compañías de seguros más importantes del mundo usan mainframe.

2.1.2 RAS: Características de los mainframes

Este concepto es muy propio de los grandes sistemas corporativos mainframe, las siglas significan **Reliability, Availability, Serviceability.**

- **Reliability:** En un mainframe se llevan a cabo extensivos autodiagnósticos, la propia máquina se diagnostica mientras está funcionando y autodetecta diferentes fallos que pudieran ocasionar un mal funcionamiento del sistema. Además de detectarlos, la propia máquina es capaz de notificarlos y corregirlos ya que cuenta con capacidades de autorrecuperación.

- **Availability:** Es el tiempo en el que el sistema está funcionando, es decir, el tiempo que está proporcionando servicios. Cuando el sistema tiene fallos es capaz de aislar las partes afectadas y seguir funcionando con una capacidad limitada. La alta disponibilidad se define como el porcentaje del tiempo en que la maquina está funcionando respecto al tiempo estimado de funcionamiento. La alta disponibilidad de un mainframe se asegura con un porcentaje de disponibilidad del 99.999%, conocido como "los 5 nueves".

- **Serviceability:** Es la rapidez o la sencillez que tiene el sistema de ser reparado o mantenido. Gracias a las excelentes capacidades de autodiagnóstico y autorecuperación del mainframe, esta máquina es muy sencilla de mantener. La propia máquina puede detectar una avería antes de que el componente en cuestión falle y se venga abajo causando un problema mayor. En vez de eso, la máquina detecta dicha avería y lo notifica al servicio técnico para que pueda reparar el fallo antes de que ocurra. Esta capacidad es una de las razones por las que un mainframe asegura la alta disponibilidad.

Respecto a la **escalabilidad,** los mainframe de IBM se pueden escalar dentro de la propia máquina añadiendo más chips de procesadores al mainframe. Un chip de procesadores o PU (*Processing Unit*) contiene procesadores y memoria. Esto quiere decir que si el mainframe va algo justo de capacidad de carga, siempre se puede aumentar su capacidad de procesamiento inyectando más hardware a la máquina para solventarlo.

Cabe destacar el concepto de LPAR a la hora de escalar un mainframe. Una LPAR es una partición lógica del sistema mainframe. Cada LPAR es capaz de operar independientemente gracias a que posee procesador, memoria y su propia instancia de sistema operativo.

Las LPAR son capaces de comunicarse con otras LPAR aun estando en diferentes sistemas mainframe. Esta característica es muy interesante a la hora de paralelizar y de cumplir con los objetivos que proponen las siglas RAS.

La clusterización se implementa utilizando lo que se conoce como Sysplex Parallel, varios mainframe se pueden “clustear” para sacar una capacidad mayor de procesamiento entre muchos sistemas que la máxima capacidad desplegada por una sola LPAR.

En los Anexos puede encontrar la historia del Sysplex Parallel. Anexo [A].

2.2 El sistema operativo z/OS

2.2.1 El sistema operativo del mainframe

Empezaremos describiendo al z/OS como un sistema operativo de 64 Bits, desarrollado principalmente para procesar grandes cantidades de trabajo (como computo de operaciones de E/S intensivas o procesamiento de grandes cantidades de trabajos o "Jobs") por un gran número (miles) de usuarios concurrentes en un entorno fiable y seguro, capaz de ejecutar aplicaciones críticas con estas garantías.

Estas características son necesarias para ofrecer un rendimiento a la altura de las situaciones más exigentes a las que un negocio se pueda enfrentar como desarrollo, compilaciones, procesamiento y tratado de ficheros de gran volumen y los diferentes servicios que un negocio desee ofrecer. Todo ello lanzado por miles de usuarios conectados concurrentemente trabajando en la misma máquina. Si desea saber más sobre la potencia de las aplicaciones mainframe en z/OS consulte [18] y [19].

2.2.2 Compatibilidad

Una característica relevante es que los programas más antiguos, compilados con arquitecturas de los años 70, siguen funcionando en esta máquina a día de hoy y sin necesidad de recompilar. Es decir, el z/OS tiene una compatibilidad hacia delante que hoy en día no tiene casi ningún entorno del mundo.

2.2.3 Capacidades de multiprogramación y multiprocesamiento

El z/OS cuenta también con capacidad de multiprogramación, es decir, es capaz de ejecutar varios programas concurrentemente. También cuenta con capacidades de multiprocesamiento, lo que permite que dos o más de sus procesadores compartan varios recursos hardware, tanto memoria como dispositivos de almacenamiento externos. Esta característica es también conocida como SMP (Symetric Multi Processing).

En Anexos puede encontrar más información acerca del hardware que el z/OS es capaz de controlar. Anexo [B].

2.2.4 z/OS: Organización de Memoria

z/OS proporciona a cada usuario un espacio de direcciones único y mantiene la distinción entre los programas y los datos representados en el espacio de direcciones. Cada programa/usuario está representado por un espacio de direcciones, es decir, cada usuario tiene una cantidad limitada de almacenamiento privado (Figura 2-1).

Figura 2-1: Cada usuario del z/OS posee un espacio de direcciones propio

Los espacios de direcciones se crean para cada usuario que inicia sesión en z/OS. También para cada proceso por lotes "Batch" se crean sus espacios de direcciones cuando estos procesos se ejecutan en el z/OS.

El z/OS gestiona el almacenamiento de tres formas diferentes: Almacenamiento real o central (CSTOR), almacenamiento auxiliar y almacenamiento virtual.

El **almacenamiento central** se encuentra dentro del chip de procesadores, dentro de la PU (Processor Unit). Esta memoria es RAM y caché, que tienen tiempo de lectura y escritura bastante altos a pesar de su escaso tamaño comparado con discos ópticos.

El **almacenamiento auxiliar** se encuentra fuera de los chips de procesadores, suelen ser discos ópticos de lectura y presentan más capacidad de almacenamiento pero mayor tiempo de lectura y escritura.

El **almacenamiento virtual** es la manera de proporcionar al sistema más cantidad de memoria principal de la que en realidad posee. En el caso del z/OS, es creada a través de la gestión de almacenamiento real y almacenamiento auxiliar a través de tablas de direcciones de memoria. El rango de almacenamiento virtual direccionable disponible para el usuario/programa/sistema operativo es un espacio de direcciones.

Al ejecutar un programa, las porciones del mismo que se están ejecutando se mantienen en almacenamiento real mientras que el resto se mantiene en almacenamiento auxiliar.

Si desea profundizar sobre la gestión del almacenamiento virtual en z/OS recomendamos la lectura del Anexo [C].

2.2.5 z/OS: Gestión interna

En el interior del z/OS existen una serie de *instrucciones* que controlan el funcionamiento del sistema:

- Una secuencia de instrucciones que ejecutan habitualmente funciones del sistema son lo que llamamos en z/OS macros o 'system programs'.
- A un conjunto de instrucciones del sistema relacionadas entre sí se le denomina módulo.
- Al conjunto de módulos que están relacionados entre sí se le denomina componente.

El z/OS se organiza internamente valiéndose de módulos, los programas del sistema (macros) y componentes del sistema.

Durante la ejecución de un programa, el z/OS utiliza una palabra de estado del programa (PSW o Program State Word). La PSW incluye unos bloques de control que contienen la información sobre el sistema, recursos y tareas.

La PSW controla el orden en el que las instrucciones son introducidas al procesador, e indica el estado del sistema en relación con el programa en curso.

Dos de los *componentes* más importantes del z/OS son: el WLM (Workload Management), que es el componente que se encarga de controlar los recursos del sistema de manera eficiente y el RTM (Recovery Termination Manager), que se encarga de la recuperación del sistema.

El **Workload Management** gestiona tanto el procesamiento de cargas de trabajo y el uso de los recursos del sistema, como los procesadores y el almacenamiento para lograr los siguientes objetivos:

- **Logro de metas**: Alcanzar los objetivos de negocio que se definieron en la instalación mediante la asignación automática de recursos sysplex (asignar capacidad de procesamiento a las cargas de trabajo en función de su importancia).

- **Rendimiento**: Lograr un uso óptimo de los recursos del sistema desde el punto de vista del sistema (rendimiento).

- **Tiempo de respuesta**: Lograr un uso óptimo de los recursos del sistema desde el punto de vista de espacio de direcciones individuales.

A continuación vamos a explicar las herramientas de las que dispone el WLM para realizar esta tarea de supervisión, sobre todo aquellas que hacen posible que el z/OS tenga la capacidad de **multiprogramación.** En el Anexo [D] se detalla el funcionamiento de este componente.

2.2.6 z/OS: Gestión interna - Interrupción de procesamiento

Una interrupción es un evento por el que se altera la secuencia en la que el procesador ejecuta instrucciones, esta interrupción puede ser planeada o no planeada.

Cuando se produce una interrupción, el hardware almacena información del programa que se interrumpió gracias a la PSW (Program Status Word) que antes habíamos mencionado y transfiere el control a la rutina apropiada para que atienda la interrupción.

En el Anexo [E] se detalla el tratamiento de interrupciones y como se realiza el cambio de contexto entre programas.

2.2.7 z/OS: Gestión interna – Creación y manejo de Unidades de trabajo

z/OS permite la creación de unidades de trabajo. Estas unidades de trabajo están representadas por dos tipos de bloques de control ("control Blocks"):

- Los **TCBs** (Task Control Blocks): Representan tareas que se están ejecutando dentro de un espacio de direcciones y corresponden a programas de usuario. Se crean con la macro ATTACH.
- **SRBs** (Service Request Blocks): Son peticiones para ejecutar una rutina de servicio del sistema. Se crean cuando un espacio de direcciones detecta que un evento afecta a otro espacio de direcciones, proporcionando un mecanismo de comunicación entre espacios de direcciones. Se crean con la macro SCHEDULE y tienen dos tipos de prioridades: local y global (más prioritaria).

Los SRBs sólo pueden crearse por programas que corren en modo supervisor, ya que tienen prioridad más alta sobre cualquier TCB en su espacio de direcciones.

Por último, en relación a las unidades de trabajo, tenemos que decir que pueden ser de dos tipos dependiendo del modo en el que se puedan interrumpir:

- **Non-preemtable**: Esta unidad de trabajo puede ser interrumpida, pero debe ser atendida una vez que se haya atendido la interrupción. Esto es un ejemplo de SRB.
- **Preemtable**: Si es interrumpida, devuelve el control de la ejecución al sistema operativo cuando la interrupción ha sido atendida. Esto es un ejemplo de TCB.

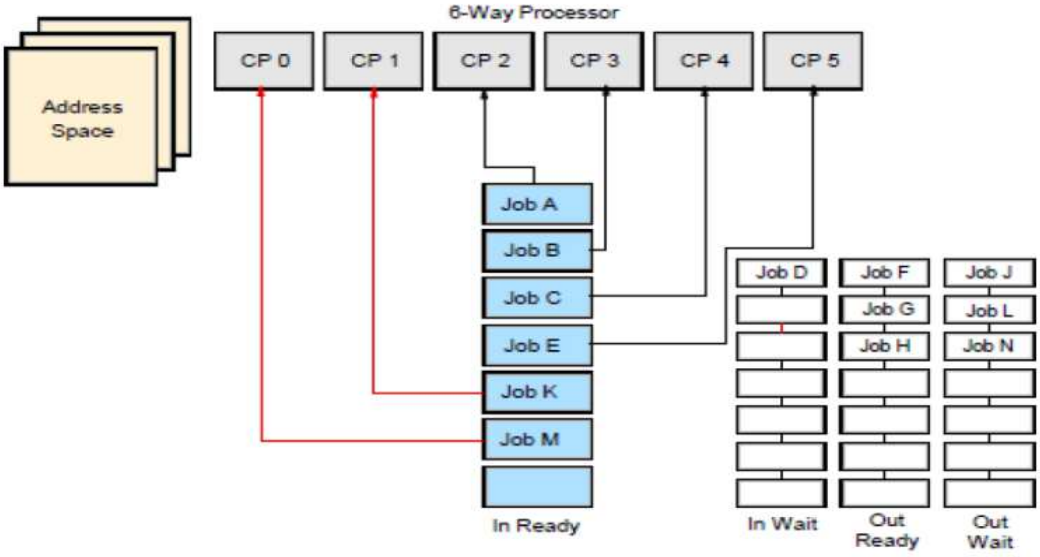
z/OS utiliza el despachador de trabajos (Job Dispatcher) para atender a la unidad de trabajo con la prioridad más alta que se encuentre en la cola de trabajo y lista para ser ejecutada.

Las unidades de trabajo se guardan en colas a la espera de ser despachadas por el Job Dispatcher, estas colas de espera pueden ser de 4 tipos:

- IN-READY: Contiene las unidades en almacenamiento central y que están a la espera de ser ejecutados.

- IN-WAIT: Contiene las unidades que se encuentran en almacenamiento central, pero se encuentran a la espera de algún tipo de evento.
- OUT-READY: Contiene las unidades listas para ejecutar que están fuera de la memoria.
- OUT-WAIT: Contiene las unidades que se encuentran fuera de memoria y a su vez están esperando algún evento.

Se puede deducir de esto que sólo las unidades de trabajo que se encuentren en la cola del tipo “IN-READY” podrán ser atendidas, estas colas se enumeraron en el orden en el que tienen que ir pasando las unidades de trabajo (figura 2-2).



La forma en la que una unidad de trabajo es tratada por el Job Dispatcher varía dependiendo del tipo de unidad de trabajo. La atención a las unidades de trabajo pueden ser de tres tipos y se producen siguiendo el orden siguiente:

- Special exits: Son las unidades de trabajo con más prioridad de todas, son de carácter especial y suelen ser rutinas invocadas por el sistema operativo en situaciones especiales.

- SRBs que cuentan con prioridad Global.

- Unidades de trabajo en espacios de direcciones que estén listos de acuerdo a la prioridad establecida de cada uno.

2.2.8 z/OS: Gestión interna – serialización y administración de los recursos

Para permitir un multiprocesamiento, es necesario la coordinación del acceso a los recursos del sistema. El componente encargado es el Global Resource Serialization, que es el que procesa las peticiones de los recursos de los programas que están corriendo en el z/OS y serializa el recurso para proteger su integridad.

Cuando un programa solicita el acceso a un recurso, indica el tipo de acceso que desea hacer: un acceso exclusivo (si se va a modificar el recurso) o un acceso compartido (si no lo va a modificar). Si el recurso no está disponible en el momento de la solicitud, el sistema suspende el programa hasta que el recurso se encuentre disponible. Cuando el recurso ya no es requerido por el programa, éste lo libera para poder ser reutilizado.

Cuando se utiliza el bloqueo de recursos, existe riesgo de deadlock (interbloqueo), es decir, que un programa necesite un recurso que tiene bloqueado otro programa. Esto ocurre, por ejemplo, si el programa 1 necesita el recurso A (recurso bloqueado por el programa 2) y a su vez, el programa 2 necesita el recurso B (recurso bloqueado en este caso por el programa 1). Los dos programas necesitan recursos para proseguir y quedan bloqueados, esto no ocurre en z/OS porque se utiliza el mecanismo de **Locking**.

Recomendamos la lectura del anexo [F] donde se detalla el mecanismo de Locking para el bloqueo de los recursos y evitar el deadlock.

Hasta aquí nuestra exposición sobre las características del z/OS. En esta sección hemos expuesto las piezas clave de las que se vale el sistema operativo para llevar a cabo tareas tan complicadas como:

- Procesar gran cantidad de trabajos en lote concurrentes con balanceo de carga automático.
- Manejar configuraciones grandes de E/S que incluyen miles de drivers de disco, librerías de cinta, impresoras o redes de terminales.
- Incorporar seguridad a aplicaciones, recursos y perfiles de usuarios.
- Usar espacio de direcciones para asegurar el aislamiento de áreas privadas.
- Asegurar la integridad de los datos, a pesar de la gran población de usuarios que un mainframe puede albergar.
- Proporcionar facilidades de recuperación extensivas, haciendo que el sistema necesite ser reiniciado muy pocas veces.

2.3 La programación en mainframes z/OS

2.3.1 El Job Control Language o JCL

Es un lenguaje de programación empleado para la ejecución de programas y procesos en entornos mainframe. Es específico para cada sistema operativo y sus instrucciones, también denominadas "pasos", son declaraciones con las que se le indica al sistema operativo las tareas a realizar, en qué orden han de ejecutarse y donde están ubicadas las fuentes de datos que necesitan.

Los programas JCL pueden usarse para multitud de propósitos. Uno de los más usados es para definir cadenas de procesamiento de datos por lotes "Batch". El z/OS proporciona las herramientas necesarias para manipulación de datos, permite descargar tablas DB2 o hacer consultas para posteriormente tratarlas con programas COBOL o con los comandos que ofrece z/OS.

El z/OS proporciona a los JCL herramientas para la manipulación de datos, una de las más comunes es el SORT. Para más información, SORT se encuentra detallado en el Anexo [G].

Para compilar y ejecutar un programa en COBOL o C también lo podremos hacer desde un JCL, de hecho, en nuestro estudio vamos a ejecutar y compilar los programas que hagamos llamando a un JCL.

2.3.2 El lenguaje COBOL

En esta parte de nuestro viaje por el entorno Mainframe vamos a centrarnos en dos lenguajes de programación ampliamente conocidos y usados en todo el mundo (y en nuestro sistema Mainframe), Cobol y C, ilustrando algunas características especiales de cada uno.

COBOL (**COmmon Business-Oriented Language**, Lenguaje Común Orientado a Negocios). Es un lenguaje de alto nivel, compilado, muy usado por programadores en todo el mundo y con una gran solera a sus espaldas (1960).

Hoy en día COBOL tiene multitud de presencia en el ámbito de los negocios, sobre todo en grandes negocios que requieren una gran capacidad de procesamiento por lotes (batch). Grandes empresas y sectores de banca que disponen de sistemas mainframe utilizan COBOL para explotar al máximo las cualidades de este lenguaje de programación.

COBOL es versátil, es capaz de hacer desde hacer el típico "Hello World" hasta el más pesado de los "batch" pasando por los más modernos y actuales modelos de Cliente-Servidor de internet.

En un informe de Gartner Group de 1997 se estimaba que de los 300.000 millones de líneas de código que se estimaba que existían en ese momento, el 80% estaban escritas en lenguaje COBOL, escribiéndose 5.000 millones de líneas de COBOL nuevas cada año.

Además, en un informe posterior (2005) se estipulaba que el 75% de los datos generados por los negocios estaban procesados por programas escritos en lenguaje COBOL.

Mientras tanto, a día de hoy, dado el número tan impresionante de programas escritos en COBOL que se encuentran operativos, la programación en lenguaje COBOL es uno de los negocios más rentables dentro del mundo de la informática.

Aunque mucha gente piense que está acabado, la realidad pinta de una manera completamente diferente, COBOL está vivo, ¡vaya si lo está!. Es líder en líneas de código en el campo de los negocios, en banca es fortísimo. A día de hoy, se sigue impartiendo en docencia en multitud de instituciones dada la demanda de este lenguaje y gracias a las decenas de miles de usuarios que siguen desarrollando en él. COBOL sigue estando soportado y evolucionando a día de hoy en parte gracias a la amplia difusión que tuvo en el pasado este lenguaje de programación.

"No sé qué lenguajes habrá en el futuro, pero seguro que Cobol estará todavía allí"
Bill Gates.

Tras este breve repaso a la historia de COBOL, vamos a enumerar algunas de las características que han hecho de COBOL uno de los lenguajes más usados del mundo:

1. **Experiencia y Fiabilidad**

   - Con sus más de 50 años de experiencia en activo, COBOL ha demostrado por sí solo su fiabilidad como lenguaje de programación.
   - El amplio espectro de negocios en los que se usa es abrumador, sobre todo en negocios que necesitan procesamiento de grandes volúmenes de datos. COBOL ha demostrado que esto no es un problema para él, ya que fue ideado para tales propósitos.

2. **Capacidad de portabilidad y compatibilidad**

   - Su capacidad de funcionamiento en casi todas las plataformas han dotado a COBOL de una magnífica portabilidad, esta es la principal razón por la que en el ámbito de los negocios COBOL está tan extendido. Fue creado en una época en la que las diferentes arquitecturas y modelos de plataformas hardware dificultaban mucho la portabilidad de aplicaciones y programas. Por ello COBOL tuvo tanta difusión.
   - COBOL surgió, entre otras cosas, con la intención de acabar con el problema de compatibilidad y portabilidad de los programas. Por ello es tan sencilla la migración de sistemas COBOL a nuevas plataformas hardware más potentes y sin necesidad de modificar el código. Esto se traduce en dinero, ya que la inversión hecha en un software construido

en COBOL no se pierde por el cambio de plataforma. Este hecho aporta a los programas COBOL una gran longevidad de funcionamiento.

3. **Autodocumentación**

   - COBOL es un lenguaje con unas características de autodocumentación muy importantes que permite a un desarrollador COBOL documentar el programa en el mismo código con características especiales, además, su sintaxis tan parecida al inglés permite entender el programa con facilidad "leyendo" simplemente el código.

4. **Es fácil de mantener**

   - Esta característica casi es una consecuencia de lo que hemos expuesto anteriormente. Gracias a la similitud con el lenguaje ordinario (inglés), un desarrollador ajeno al creador del programa es capaz de entender con facilidad el código y así poder modificarlo, mejorarlo o simplemente continuar el trabajo donde otro lo dejó, evitando así el tener que desechar el programa.

2.3.3 El lenguaje C

Este lenguaje de programación fue desarrollado entre los años 1969 y 1973, aunque el periodo en el que más se avanzó fue el año 1972. En los laboratorios Bell de AT&T fue creado por Dennis Ritchie. El nombre de "C" vino de que muchas de sus características fueron heredadas de su lenguaje predecesor, el lenguaje "B".

Fue ideado principalmente para poder construir el sistema operativo UNIX. Este lenguaje introducía una serie de modificaciones respecto a su lenguaje predecesor, el "B". Las modificaciones que aportaba C frente a B eran las uniones, estructuras de datos y arrays, en definitiva, características que hacían a C capaz de declarar estructuras de datos más complejas.

A partir de este punto C comenzó a evolucionar gracias a Brian Kernighan y a Dennis Ritchie tras publicar su famoso libro ***el lenguaje de programación C*** o conocido entonces como "*La biblia del C*" [8]. El lenguaje pasó por múltiples versiones y estandarizaciones hasta llegar a la actual versión, la C11, que es el nombre informal para la ***ISO/IEC 9899:2011*** el último estándar publicado para C. El borrador final "N1570" fue publicado en abril de 2011. El nuevo estándar superó su última revisión el 10 de octubre de 2011 y fue oficialmente ratificado por la ISO y publicado el 8 de diciembre de 2011.

Los tipos de datos con los que C contaba inicialmente eran INTEGER (Enteros), CHAR (caracteres), FLOAT (Números reales con simple precisión) y DOUBLE (Números reales con doble precisión), pero tras sucesivas revisiones añadieron los tipos de datos SHORT

(enteros de longitud menos o igual a un INTEGER), LONG (enteros con longitud mayor o igual a un INTEGER), UNSIGNED (Sin signo).

En posteriores revisiones se añadieron modificaciones como el tipo de dato STRUCT, enumeraciones, el identificador CONST (datos de sólo lectura) y entre otros cambios, añadieron diversas modificaciones en la forma de definir el código como, por ejemplo, la forma de la declaración de las funciones. También proporcionaron la capacidad de incluir a C multitud de librerías estándar, característica que ha sido vital en su amplio uso.

Sus características…

C es un lenguaje compilado, está clasificado entre medio y bajo nivel. Posee unas funcionalidades extensibles más que destacadas gracias a la multitud de librerías capaces de convertirlo en un lenguaje de programación muy completo y versátil. Además, con muy pocas modificaciones de código presenta muy buena portabilidad hacia una gran variedad de entornos.

El lenguaje C en general es polivalente y está muy aceptado como un lenguaje para propósito general, es estructurado y presenta actualmente una gran cantidad de definiciones y tipos de datos.

C en general es eficiente gracias a que su compilador es muy ligero y genera poco código que no sea de utilidad. También esto es gracias a que C es de bajo nivel.

C presenta quizás una excesiva libertad a la hora de escribir código, esto es un arma de doble filo. El programador se siente bastante cómodo, pero desgraciadamente en la práctica esto se suele traducir en errores de programación que a veces, por ser correctos sintácticamente no se detectan en tiempo de compilación.

Otra cosa de la que C no está dotado es de funciones propias para el manejo de **entrada y salida** (las importa de librerías). Tampoco dispone de funciones propias para el manejo de cadenas de texto (Strings) algo esencial para un manejo eficiente de los datos. Este trabajo se dejaría en manos de las librerías, lo que podría suponer un problema de portabilidad.

Respecto a su estructura, debemos decir que como todo lenguaje, posee palabras reservadas para identificar instrucciones y estructuras. Estas palabras reservadas deberán escribirse en minúscula, ya que a diferencia de COBOL, el lenguaje C diferencia entre mayúsculas y minúsculas.

Vamos a ver la estructura que seguiría un programa C:

1. **llamadas a librerías**
2. **declaración de funciones (prototipos de funciones definidas)**
3. **declaración de variables globales**
4. **main()**
5. **{**

6. **declaración de variables locales**
7. **sentencias**
8. **}**

Como vista general, esta es la estructura que tiene un programa escrito en lenguaje C. Primero se escriben las llamadas a las librerías que vamos a utilizar (tanto del sistema como definidas por nosotros), posteriormente las funciones que nos hemos declarado para modularizar nuestro programa.

Unas buenas prácticas de programación de C consisten en agrupar las funciones por funcionalidad en distintos ficheros con extensión .c, separándolas de nuestro programa principal o 'main' (aunque se pueden definir en el propio archivo que contiene el programa principal (ver estructura programa C, páginas 14-15) para después declararlas en su correspondiente archivo .h. Así podremos incluir los ficheros .h en nuestro programa principal en el apartado anterior de llamadas a librerías permitiéndonos así crear nuestras propias librerías con nuevas funciones y definiciones de tipos de datos.

Generalmente en los archivos .h se incluyen las definiciones de las funciones o "cabeceras", las definiciones de tipos de datos y las definiciones de estructuras que vayamos a utilizar. En el .c se implementa el propio código de las funciones, separando de esta manera en 'módulos' nuestro programa o aplicación, quedando un archivo .c correspondiente al programa principal y a parte, las parejas de archivos .c y .h de nuestras funciones definidas. Esto queda a la decisión del programador de cómo **modularizar** su programa.

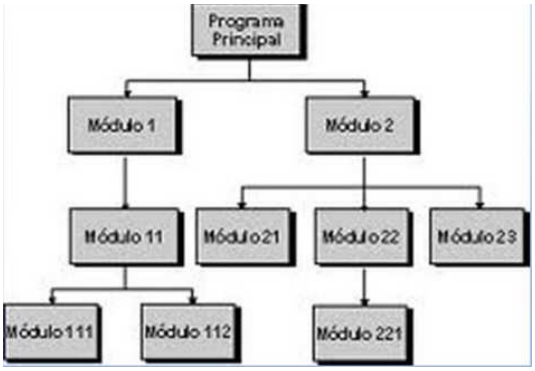


Figura 2-2: Ejemplo de programación modular

La estructura de un programa C es simple, tenemos un programa principal 'main' (ver estructura programa C, páginas 14-15) en el que se encuentran las operaciones, instrucciones y declaraciones de variables que vamos a usar en nuestro programa. Respecto a las sentencias (ver estructura programa C, páginas 14-15) son todas las instrucciones que vamos a realizar en nuestro programa. Respecto a variables (ver estructura programa C, páginas 14-15), hay que destacar que existen variables globales y locales, las variables globales se declaran fuera de cualquier función y son visibles desde cualquiera de ellas (incluida la 'main'). Las variables locales sólo pueden ser utilizadas desde las funciones que las crearon.

2.3.4 Data Base 2 (DB2)

En este apartado vamos a destacar las características de DB2, el sistema de gestión de bases de datos de IBM.

DB2 es un sistema para administración de bases de datos relacionales "RDBMS" multiplataforma, especializado en trabajar en ambientes distribuidos. Esto quiere decir que muchos usuarios de forma local comparten una misma información centralizada. Algo que para los negocios es fundamental a la hora de trabajar en ambientes distribuidos con diferentes localizaciones.

Integridad y seguridad
DB2 asegura integridad para sus datos, aún en un punto de colapso del sistema. También ofrece seguridad con distintos grados de granularidad para cada rol de usuario.

En relación al nivel de granularidad de seguridad de los datos, cabe destacar que se basa en dos sistemas para la restricción de permisos a varios niveles. Uno de ellos (DAC, Discretionary Access Control) suele estar asignado por defecto a la seguridad de tablas y el segundo de ellos (LBAC: Label-Based Access Control) suele estar asignado a nivel de filas y columnas, pero es perfectamente configurable para asignarlo a nivel de tablas y así sustituir al DAC si así se prefiriera.

Esta capacidad de intercalar estos dos sistemas de seguridad hace posible ofrecer un nivel más de libertad a la hora de proporcionar diferentes políticas de control para el acceso a los datos y tablas.

Otra característica a destacar en relación a la seguridad es los llamados 'Trusted Context' (contextos de confianza), introducido en la versión 9 del DB2. Con esta tecnología se es capaz de incluir direcciones IP correspondientes a 'puentes' entre diversas plataformas de datos que incluso pueden tener sistemas de verificación de autenticidad/seguridad diferentes. Con este método lo que se pretende conseguir es aumentar el rendimiento evitando las comprobaciones de autenticidad entre las entidades.

Escalabilidad, universalidad y flexibilidad
DB2 es capaz de trabajar desde en ordenadores personales hasta en grandes sistemas mainframe, proporcionando gran rendimiento gracias a su capacidad única de escalabilidad. En el ámbito transaccional es capaz de dar salida a múltiples necesidades de negocio como, por ejemplo, procesar transacciones de misión crítica (denominadas OLTP) y hacer un análisis minucioso de los datos para dar soporte a la toma de decisiones en cuanto a balanceo de carga, rendimiento y funcionalidad. Recomendamos la lectura de las referencias [13] y [14].

DB2 además pretende ser una base de datos universal y también multiplataforma, es capaz de funcionar en 16 tipos de plataformas, de las cuales 10 no son de IBM, brindando soporte a un amplio tipo de clientes (figuras 2-4 y 2-4). Respecto a los servicios de internet, contenido multimedia y demás, DB2 está más que preparado dado que permite almacenar cualquier tipo de datos definido previamente por el usuario, ya sea audio, video, texto, etc.

Figura 2-3: Instalación DB2 en equipo Windows

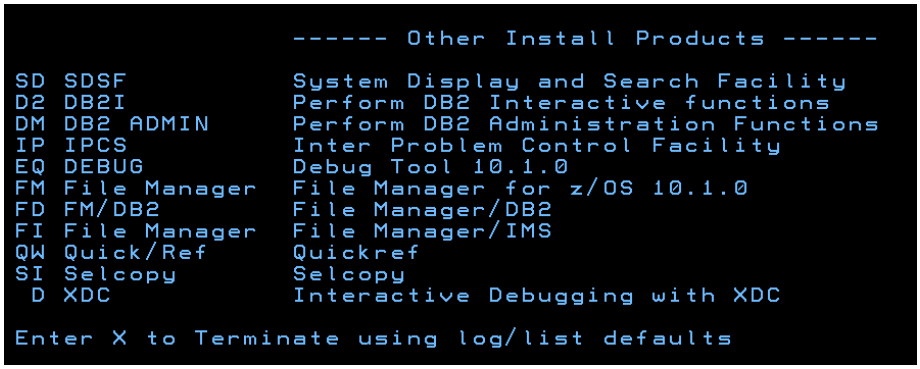


Figura 2-4: Instalación de DB2 en entorno mainframe

Business intelligence

DB2 permite la mejora del rendimiento gracias al uso de datos activos, DB2 es capaz de trabajar con cientos de clientes conectados accediendo desde la web.

DB2 abarca desde tipos de datos básicos hasta tipos de datos complejos y definidos por el usuario como el audio, el video y el contenido multimedia tan demandados por las aplicaciones de e-bussines. Puede consultar las referencias [17] y [18] para saber más acerca de aplicaciones y servicios web en el mainframe.

DB2, mantenimiento automático

DB2 posee unas características de automatización potentes, esto permite al usuario en cierta manera 'despreocuparse' de tareas como, por ejemplo:

- Copia de seguridad automática: El sistema permite a DB2 hacer copias de seguridad de la base de datos periódicamente. Esto permite al usuario despreocuparse tanto de que la base de datos haga copias de seguridad regular y

correctamente, como de conocer la sintaxis de comandos necesarios para hacer los backup.

- Actualización de Estadísticas: La recopilación automática de estadísticas permite a DB2 mejorar el rendimiento de la base de datos gracias a la actualización automática de las estadísticas de las tablas. Esto ayuda al optimizador a elegir en cada caso la solución más adecuada en cada momento para cada consulta de cada tabla.
- Autoajuste del uso de la memoria: DB2 tiene habilitado por defecto el autoajuste de memoria. Esta función permite a DB2 configurar autónomamente los valores de los parámetros de configuración de la memoria. Si está habilitada esta funcionalidad, el administrador de la memoria distribuye los recursos de forma automática y dinámica a los consumidores de memoria que lo requieran.
- DB2 también posee capacidades de detección de carga de trabajo dinámicas, permitiendo así la gestión automática del rendimiento distribuyendo dinámicamente la memoria entre los procesos para la optimización del rendimiento del sistema.

Consulte la referencia [15] para más información acerca del mantenimiento y la administración del DB2.

Compresión

Esta característica es muy interesante en las bases de datos, dado que cuando se disponen de volúmenes de datos realmente grandes esto puede suponer un problema de almacenamiento.

Esta característica en bases de datos relacionales es complicada de implementar dado que el almacenaje de los datos se basa en datos por fila. La complicación viene en cuanto a que una misma fila de datos (registro) está formada por varios tipos de datos como float, integer o alfanuméricos. Estos tipos de datos para comprimirlos óptimamente utilizan desgraciadamente diferentes algoritmos de compresión. Esto significa que si en la realidad se comprimieran los datos por filas, el algoritmo a utilizar debería de venir determinado por el común denominador de todos los campos.

El método de compresión utilizado por IBM se denomina **Tokenstation** y su funcionamiento se basa en la búsqueda de patrones repetidos en los datos. Cuando esta casuística aparece se crea un símbolo que lo contendrá en una tabla con el formato dato-símbolo, cuando se haga referencia a este símbolo, DB2 sabrá que debe irse a la tabla dato-símbolo para sustituir el símbolo por su valor verdadero en la tabla.

PureXML

DB2 ofrece un ambiente híbrido para datos relacionales y XML de tal forma que integra XML de manera nativa, lo que IBM denomina PureXML.

Esta tecnología permite guardar documentos del tipo de datos XML para poder realizar búsquedas de manera jerárquica e integrarlo con búsquedas relacionales mezclando ambos tipos de datos, XML y relacionales. Si desea saber más Acerca de PureXML y el DB2 en el z/OS consulte la bibliografía [11] y [12] del apartado de referencias de este documento.

3 Diseño

3.1 Creación de programas en el Mainframe

En este apartado vamos a definir lo que queremos hacer, analizando lo que necesitamos para poder realizar las distintas pruebas que nos propusimos al principio de este estudio.

Recordemos que nos propusimos hacer una comparativa entre el lenguaje C y el lenguaje COBOL. Adicionalmente, también vamos a testear el DB2 del mainframe a la vez que ilustramos que en el lenguaje COBOL se puede embeber código SQL y de esta manera, llamar al DB2.

3.1.1 ISPF, TSO y RDZ

Para poder crear un programa en el mainframe primero tendremos que saber cómo hacerlo. La manera más frecuente es acceder al mainframe mediante una sesión en el TSO y usar las herramientas que el ISPF nos proporciona. Empezaremos definiendo TSO e ISPF.

TSO (Time Sharing Option): Es un entorno interactivo de tiempo compartido que proporcionan los sistemas operativos de mainframe de IBM.

Tiempo compartido significa que el usuario no percibe que otros usuarios están conectados al mismo tiempo al sistema operativo, dando la impresión de que el usuario es el único que esta “controlando” el mainframe.

ISPF (Interactive System Productivity Facility): Es un conjunto de herramientas que se proporciona a los sistemas operativos mainframe de IBM.

Principalmente proporciona una interfaz de usuario con menús para ejecutar herramientas del sistema bajo el TSO. Consulte la bibliografía [4] y [10] para saber más sobre el ISPF y el TSO.

RDZ (Rational Developer For System Z**)**: Se trata de un IDE (Integrated Developer Enviroment o Entorno de Desarrollo Integrado) capaz de permitirnos desarrollar aplicaciones y proyectos en un entorno mainframe de IBM. Este IDE abre una sesión TSO con el mainframe que queremos conectar, ofreciendo las herramientas de las que el ISPF dispone con una perspectiva diferente y más conocida a los desarrolladores más modernos.

También ofrece facilidades para desarrollos de aplicaciones en entorno mainframe, como esqueletos de JCL de compilación.

Una vez definidos estos conceptos podremos empezar a conectarnos al mainframe para codificar nuestros programas. En este estudio encontraremos referencias tanto al ISPF como al IDE Rational Developer for System (RDZ). Consulte la bibliografía [5] y [21] para saber más sobre el RDZ.

3.1 Comparativa de lenguajes C – COBOL

3.1.1 Comparativas de lenguajes

Como adelantamos al comienzo de nuestro estudio, vamos a hacer una comparativa de estos dos grandes lenguajes de programación. Veremos las similitudes que guardan y las diferencias que esconden. Para hacer esto un poco más real, comentamos que íbamos a hacer las pruebas en un entorno mainframe real, el Marist College.

La primera característica que vamos a destacar de COBOL frente a C es el retorno de las funciones para el tratamiento de los ficheros. Recordemos que COBOL es un lenguaje de alto nivel, esto se traduce en facilidades para el programador a la hora de desarrollar aplicaciones de negocio. Una de las características que esto implica es la automatización de la gestión de memoria en COBOL (No existen punteros en COBOL). También hay que destacar la información que proporcionan las funciones, que es el caso que vamos a describir a continuación.

Primeramente nos vamos a centrar en la función de apertura de ficheros, la función para la apertura de un fichero en C es la siguiente:

```
FILE *fopen(const char *nombre, const char *modo)
```

En donde *const char *nombre* representa al nombre del fichero en cuestión y *const char *modo* representa la modalidad con la que el fichero se va a abrir, como solo lectura, o lectura y escritura, etc.

Respecto al retorno de la función observamos que se trata de un puntero de tipo FILE (Fichero), Si la apertura del fichero fue exitosa, esta función devolverá un puntero al fichero abierto, en caso de no lograr abrir el fichero, nos retornará un valor nulo o NULL.

En COBOL la función homóloga sería la siguiente:

```
OPEN modo nombre
```

En donde *nombre* indica el nombre del fichero y *modo* indica el método de apertura del fichero. Puede ser en modo lectura, escritura o más modalidades que no afectan a nuestro ejemplo.

El retorno de la función OPEN, y en definitiva de cualquier retorno de una función de tratamiento de ficheros de COBOL, viene informado en la variable asociada al FILE-STATUS del fichero, que recordemos estaba en la propia definición del fichero, en la ENVIROMENT-DIVISION (anexo [I]).

3.1.2 Batería de pruebas: programas COBOL & C

Para comparar estos dos lenguajes hemos optado por hacer una medición de tiempos a nivel de operación. Haremos dos programas equivalentes, uno en lenguaje C y otro en lenguaje COBOL.

Ejecutaremos las operaciones más comunes en el ámbito de tratamiento de datos así como las operaciones más básicas que nos ofrecen estos lenguajes, para ello, con unos simples bucles nos bastarán. Probaremos las operaciones aritméticas (suma, resta, multiplicación, división y potencia), operaciones de asignación de variables (implementaremos un Swap) y operaciones con ficheros (leer y escribir).

Para poder ejecutar estos programas en el mainframe, para cada programa será necesario un JCL con los pasos de compilación, enlazado y lanzado del programa. También, como vamos a utilizar operaciones con ficheros, en el programa COBOL hemos de añadir un paso adicional que nos borre el fichero generado si ya. El programa COBOL no nos creará el fichero, sino que lo hará el propio JCL.

3.1.3 Programas COBOL – DB2

En esta parte de nuestro estudio del mainframe vamos a probar el DB2 del mainframe, para ello nos dispondremos a hacer un total de tres programas que ejecuten cada uno un tipo de consulta. Nuestro objetivo es medir cómo se comporta el DB2 del mainframe para consultas grandes con elevado coste computacional para diferentes tamaños de tablas. Haremos la medición para 10.000, 100.000 y 900.000 registros por tabla.

La base de datos sobre la que los probaremos viene almacenada en el DB2 del mainframe, es la base de datos SAMPLE y para hacer nuestras consultas replicaremos el siguiente subesquema de esta base de datos:

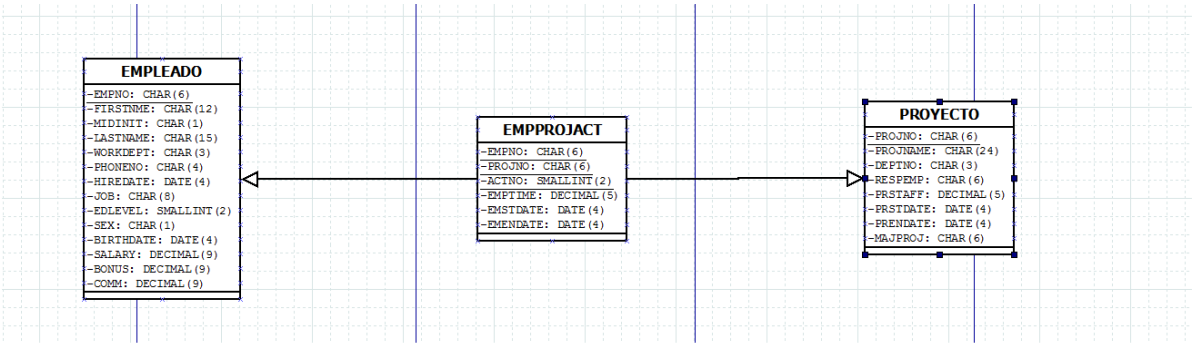


Figura 3-1: Esquema de tablas a acceder en las pruebas

El primer programa tendrá una consulta que implemente JOIN de tablas, por ejemplo, haremos un doble JOIN para unir las tres tablas de nuestro diagrama (Figura 3-1). Para el segundo programa, implementaremos una consulta en la que aparezca dos veces la cláusula NOT EXIST. Para el tercer programa probaremos cómo se comportan las funciones

agregadas, usaremos la función para el cálculo del valor medio en la consulta, también incluiremos una condición con la clausula HAVING.

Por último, necesitaremos un programa que nos inserte gran cantidad de registros, dado que en nuestras tablas no existen suficientes como para hacer las mediciones con la magnitud que queremos hacer. Los insertaremos en las tablas EMPLEADO, PROYECTO y EMPROJACT mediante el desarrollo de tres programas, uno para cada tabla.

4 Desarrollo

Hemos hecho un desarrollo para comparar a COBOL y a C en tiempos de ejecución para la mayoría de sus instrucciones. Las separaremos en instrucciones aritméticas, de asignación de variables y de utilización de ficheros. También nos propusimos ilustrar el DB2 embebido en COBOL testeando su rendimiento para consultas sobre tablas con gran volumen de registros.

A lo largo de este capítulo primero detallaremos los JCL de compilación y luego detallaremos el desarrollo de los programas.

4.1 JCL de compilación y lanzado de los programas

4.1.1 JCL de los programas de comparación de apertura de ficheros: Cobol & C

En esta parte de desarrollo vamos a crear nuestro primer JCL de compilación, lo llamaremos COBOL2J y ejecutará el programa llamado COBOL2. Para ello podemos usar RDZ para crearnos un esqueleto de JCL. Tras modificarlo logramos un JCL como el que se ilustra en el anexo [J]. A continuación detallaremos el JCL que acabamos de generar.

La compilación
El paso de compilar un programa en el JCL es el siguiente (figura 4-1). En este paso se le indican algunas librerías, entre las más importantes están: la librería donde se encuentra el programa a compilar (COBOL2), la especificamos en la SYSIN (KC02520.COBOL.SRC) y la librería (KC02520.LANG.OBJ) donde se debe depositar el programa compilado en código objeto, también lo llamaremos COBOL2.

```
000001 //KC025201 JOB ,
000002 // MSGCLASS=H,MSGLEVEL=(1,1),TIME=(,4),REGION=144M,COND=(16,LT)
000003 //*
000004 //STP0000 EXEC PROC=ELAXFCOC
000005 //* CICS=,
000006 //* DB2=,
000007 //* COMP=
000008 //COBOL.SYSLIN DD DISP=SHR,
000009 //         DSN=KC02520.LANG.OBJ(COBOL2)
000010 //COBOL.SYSDEBUG DD DISP=SHR,
000011 //         DSN=KC02520.COBOL.SYSDEBUG(COBOL2)
000012 //COBOL.SYSLIB DD DISP=SHR,
000013 //         DSN=KC02520.COBOL.SRC
000014 //COBOL.SYSXMLSD DD DUMMY
000015 //COBOL.SYSIN DD DISP=SHR,
000016 //         DSN=KC02520.COBOL.SRC(COBOL2)
000017 //*
```

Figura 4-1: Paso de compilación del programa de ejemplo COBOL2.

Como podemos ver, el proceso que se ejecuta para compilar nuestro programa es el ELAXFCOC.

El enlazado

El paso de enlazado del programa es el paso LKED (figura 4-2) y en él se llama al proceso ELAXFLNK. Se le debe especificar la librería donde se encuentra el código objeto depositado previamente por el paso de compilación. Recordemos que era en KC02520.LANG.OBJ. También se le debe especificar la librería donde guardará el programa ejecutable, en nuestro caso será KC02520.LANG.LOAD.

```
000019 //LKED EXEC PROC=ELAXFLNK
000020 //LINK.SYSLIN DD *
000021       INCLUDE OBJ0000
000022 /*
000023 //LINK.SYSLMOD    DD  DISP=SHR,
000024 //         DSN=KC02520.LANG.LOAD(COBOL2)
000025 //LINK.OBJ0000 DD DISP=SHR,
000026 //         DSN=KC02520.LANG.OBJ(COBOL2)
```

Figura 4-2: Paso de LINK del programa de ejemplo COBOL2.

La ejecución

El paso de ejecución del programa (figura 4-3) es el llamado GO y el proceso que invocamos es ELAXFGO, a este paso se le debe especificar tanto la información que necesite el programa, como los parámetros y los ficheros. También se le debe indicar la librería donde se depositó el ejecutable en el paso anterior de enlazado.

```
//******* ADDITIONAL JCL FOR LINK HERE ******
//GO    EXEC   PROC=ELAXFGO,GO=COBOL2,
//           LOADDSN=KC02520.LANG.LOAD
//SYSPRINT DD SYSOUT=*
//
```

Figura 4-3: Paso de ejecución del programa de ejemplo

Pero este no es el compilador de COBOL, los procesos ELAXFCOC, ELAXFLNK y ELAXFGO están dados de alta en el mainframe y cuando se ejecuta cada uno de ellos,

el mainframe lo transforma en el compilador de COBOL. Este proceso lo podemos encontrar en el anexo [K].

Ahora vamos a generar el JCL correspondiente al programa en lenguaje C,
La generación del JCL correspondiente al programa C sigue un proceso análogo al que acabamos de seguir, en el anexo [J] puede encontrar este JCL.

4.1.2 JCLs de los programas de batería de pruebas: Cobol & C

En esta parte de nuestro desarrollo crearemos dos JCL, el primero lo llamaremos BATECOBJ y será el JCL que compile nuestro programa de batería de pruebas para medir los tiempos de las operaciones COBOL. El segundo JCL hará lo mismo que el primero pero para un programa escrito en lenguaje C, lo llamaremos BATECJ.

Empezaremos con el JCL para el benchmark de COBOL. Para este JCL de compilación tendremos que hacer exactamente los mismos pasos que para el programa anterior, salvo con dos diferencias:

1º Añadiremos un paso de borrado del fichero que le pasemos al programa de batería de pruebas BATECOB:

```
000005 //*****************************  PASO DE BORRADO  ************************
000006 //BORRADO   EXEC PGM=IDCAMS
000007 //SYSPRINT  DD SYSOUT=*
000008 //SYSIN DD *
000009         DEL KC02520L.FILEPRU
000010         SET MAXCC = 0
```

Figura 4-4: Paso de Borrado del fichero KC02520L.FILEPRU

2º Añadiremos la creación del fichero en el paso GO del JCL:

```
000032 //*****************************  PASO DE EJECUCION  ***********************
000033 //GO     EXEC   PROC=ELAXFGO,GO=BATECOB,
000034 //            LOADDSN=KC02520.LANG.LOAD
000035 //SYSUT1   DD  DSN=KC02520L.FILEPRU,DISP=(NEW,CATLG,DELETE),
000036 //             SPACE=(CYL,1000),DCB=(RECFM=FB,LRECL=2,BLKSIZE=0),
000037 //             UNIT=SYSDA
000038 //SYSPRINT DD SYSOUT=*
000039 //
****** ***************************** Bottom of Data ****************************
```

Figura 4-5 Paso de ejecución del programa con la creación del fichero KC02520L.FILEPRU

Consultar la referencia [2] para ver cómo hemos alocado este fichero desde el JCL.
La generación del JCL correspondiente al programa C sigue un proceso análogo al que seguimos en el apartado anterior con un ligero cambio. A la hora de generar el esqueleto con el RDZ, indicaremos el fichero que queremos que se cree en el programa. En el anexo [J] puede encontrar estos JCL.

4.1.3 JCL de los programas de pruebas: Cobol - DB2

Hemos desarrollado seis JCL, tres de ellos son para los programas de medición de tiempos de las consultas. A estos JCL los llamaremos como a los programas que lanzan pero con una 'J' al final:

- SELCRUZJ: JCL que construye y lanza el programa que mide los tiempos al ejecutar un doble cruce.
- SELNOTJ: JCL que construye y lanza el programa que mide los tiempos al ejecutar un doble NOT EXISTS.
- SELAVGJ: JCL que construye y lanza el programa que mide los tiempos al ejecutar una consulta que implementa la función agregada para el cálculo de la media (AVG).

Los otros tres JCL restantes son para ejecutar los tres programas que ejecutan inserciones masivas a las tablas:

- INSPROJJ: JCL que construye y lanza el programa que inserta en la tabla PROYECTO.
- INSEMPJ: JCL que construye y lanza el programa que inserta en la tabla EMPLEADO.
- INSEMACJ: JCL que construye y lanza el programa que inserta en la tabla EMPROJACT

La generación de estos JCL es común entre ellos, la haremos de manera análoga a los vistos anteriormente, solo que con algunas modificaciones.

1.- Precompilación

Con la ayuda de RDZ logramos para que nos genere el esqueleto de un JCL para un programa con acceso a DB2. Una vez hecho esto, introduciremos un paso previo a la compilación, el **precompilador** de DB2 (figura 4-6).

El precompilador de DB2 se encarga de separar las sentencias SQL que tengamos en nuestro programa. Este código SQL lo convierte en llamadas a funciones que se comunican finalmente con el DB2 para ejecutar las consultas y los mandatos DB2 de nuestro programa. Por otra parte, el precompilador guardará las sentencias SQL en un modulo a parte, el DBRM o modulo de solicitud de base de datos [1].

```
000001 //KC025201 JOB ,
000002 // MSGCLASS=H,MSGLEVEL=(1,1),TIME=(,20),COND=(16,LT)
000003 //*************************** PASO DE PRECOMPILACION ******************
000004 //P0000 EXEC PROC=ELAXFCOP,REGION=0M,
000005 //           DBRMDSN=KC02520.DB2DBRM,
000006 //           DBRMMEM=SELCRUZ
000007 //COBPRE.SYSLIB DD DSN=DSNA10.SDSNLOAD,DISP=SHR
000008 //COBPRE.SYSIN DD DISP=SHR,
000009 //         DSN=KC02520.COBOL.SRC(SELCRUZ)
000010 //*
```

Figura 4-6 Paso de precompilación del programa DB2 SELCRUZ

2.- BIND

La segunda modificación que tenemos que realizar en nuestro JCL de compilación es incluir un paso de **BIND** de programa (figura 4-7). Puede encontrar este JCL en el anexo [J].

```
//***************************** PASO DE BIND ******************************
//BIND EXEC PGM=IKJEFT01
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//DBRMLIB  DD  DSN=KC02520.DB2DBRM(SELCRUZ),DISP=SHR
//SYSTSIN   DD  *
     DSN SYSTEM(DBAG)
     BIND PACKAGE(DSN81010)-
     OWNER(KC02520) -
     MEMBER(SELCRUZ) -
     ENCODING(EBCDIC) -
     LIBRARY('KC02520.DB2DBRM') -
     ACTION(REP) -
     VALIDATE(BIND)
     BIND PLAN(BENCHD) -
     ACTION(REP) -
     ENCODING(EBCDIC) -
     PKLIST(DSN81010.*)
```

Figura 4-7 Paso de BIND del programa DB2 SELCRUZ

El BIND vincula la información DB2 almacenada en el modulo DBRM a un paquete (PACKAGE de la imagen, figura 4-7). Este paquete de información DB2 posteriormente se asocia a un **Plan**. El Plan indica al programa las vías de acceso al DB2 a través de la información contenida en el DBRM.

Mencionar que los paquetes de información DB2 se pueden agrupar en colecciones de paquetes, de esta manera se puede asociar el Plan a una colección de paquetes y añadir/eliminar paquetes de información DB2 de esta colección si nuestra aplicación fuera evolucionando añadiendo mas vías de acceso a DB2 sin tener que volver a vincular el plan de nuevo a la nueva información de DB2 que se fuera generando.

Si observamos la figura 4-8 veremos todo el proceso de la generación de un ejecutable COBOL.

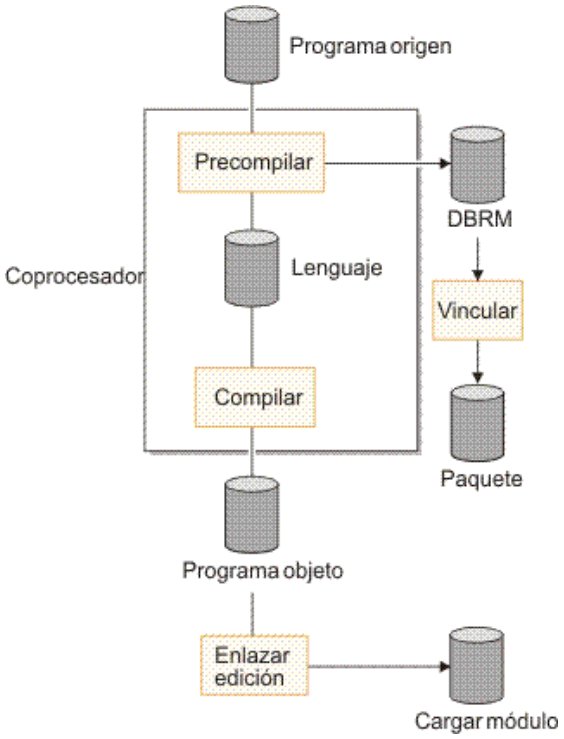


Figura 4-8 Esquema de compilación de un programa con acceso a DB2

3.- Ejecución

Ha sido necesario modificar el paso de ejecución del programa para poder implementar las llamadas a DB2 (figura 4-9).

```
//***************************** PASO DE EJECUCION ************************
//************************************************************************
//* RUN PGM
//************************************************************************
//RUNPGM   EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT),REGION=0M,
// TIME=1440
//STEPLIB  DD DSN=CEE.SCEERUN,DISP=SHR
//         DD DSN=FEK900.SFEKLOAD,DISP=SHR
//         DD DSN=KC02520.LANG.LOAD,DISP=SHR
//         DD DSN=KC02520.DB2DBRM,DISP=SHR
//         DD DSN=DSNA10.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSTSIN DD *
         DSN SYSTEM(DBAG)
         RUN PROGRAM(SELCRUZ) PLAN(BENCHD)
```

Figura 4-9 Paso de ejecución del programa SELCRUZ

La primera modificación que destacamos es la llamada a un programa llamador, el IKJEFT01. Este programa se encarga de llamar a nuestro programa ejecutable SELCRUZ y asociarle la información DB2 necesaria para su ejecución, como su plan de acceso a DB2 y el subsistema de base de datos donde se encuentra la base de datos que contiene nuestras tablas.

4.2 Programas de prueba

4.2.1 Programa Apertura de ficheros: Cobol & C

Los desarrollos que detallamos en este apartado son los programas de ejemplo de apertura de ficheros en COBOL y C. Los dos desarrollos son muy similares, se trata de intentar abrir un fichero que no exista. Para ello con la función apropiada en cada lenguaje y un sencillo control de errores conseguiremos nuestro objetivo. En caso de que no pueda abrirse el fichero, imprimirá el retorno de sendas funciones de apertura a la salida.

En el anexo [L] podemos ver ambos programas.

4.2.2 Programa batería de pruebas: Cobol & C

En este apartado describiremos los aspectos relevantes del desarrollo de los programas que miden en tiempo las operaciones de los lenguajes COBOL y C. Ambos programas son equivalentes en funcionalidad y codificación, así que sólo nos dedicaremos a ilustrar uno de ellos, el COBOL.

El programa se divide en tres párrafos principales, INICIO, PROCESO y FIN, El párrafo INICIO contiene la inicialización de variables y copys de salida, mientras que toda la lógica del programa se implementa en el párrafo PROCESO y en el párrafo FIN se imprime.

La estructura del programa es un bucle principal (figura 4-9) que itera tantas veces como operaciones queramos probar. También captura las marcas de tiempo antes y después de haber probado cada operación para poder cuantificar cuanto tardó una operación en ejecutarse. En nuestro caso, estas operaciones son: aritméticas (suma, resta, multiplicación, división y potencia), de asignación de variables (swap) y de ficheros (leer y escribir).

```
000141*****************************************************************
000142* PROCESO DE PROGRAMA
000143*****************************************************************
000144     PERFORM VARYING WK-NUM-PRUE FROM 1 BY 1
000145     UNTIL WK-NUM-PRUE > CN-NUM-TEST
000146
000147         PERFORM GET-MARCA-TIEMP
000148            THRU GET-MARCA-TIEMP-EXIT
000149
000150         MOVE WK-MARCA                    TO WK-MARCA-INI
```

Figura 4-10 Bucle principal que ejecutara cada una de las operaciones

Para cada operación (figura 4-10) la evaluamos iterándola un número grande de veces para que nos salgan marcas de tiempo apreciables. El número de veces que ejecutamos cada operación varía dependiendo del tipo de operación. En nuestro programa hemos diferenciado entre operaciones de ficheros, operaciones aritméticas y de asignación de variables, dado que suponemos que, las operaciones de ficheros tienen un mayor coste computacional que las operaciones aritméticas. El número de veces que ejecutamos cada operación lo haremos definitivo cuando realicemos las pruebas ya que tenemos que encontrar un valor adecuado para la prueba.

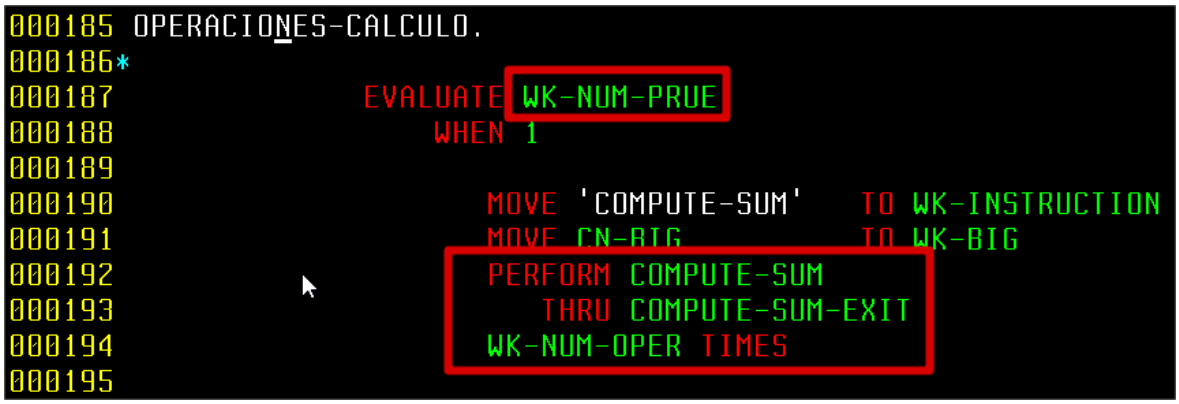


Figura 4-11 La operación suma se ejecuta tantas veces como lo indique WK-NUM-OPER

Una vez se haya terminado de iterar la operación que estamos probando, el programa capturará la marca de tiempo (figura 4-11) y calculará la diferencia entre las marcas de tiempo antes de la prueba de la operación y al terminar la prueba. Una vez se obtenga la duración, la imprimirá en un log que mostraremos por salida estándar del mainframe (SPOOL).

```
GET-MARCA-TIEMP.
*
     ACCEPT WK-MARCA-TMP FROM TIME.
*
     COMPUTE WK-MARCA ROUNDED = ( WK-MARCA-HORA * 3600 )  +
     ( WK-MARCA-MIN * 60   )  +
     WK-MARCA-SEC             +
     ( WK-MARCA-CEN / 100  )
*
     .
GET-MARCA-TIEMP-EXIT.
```

Figura 4-12 De esta manera se captura la marca de tiempo en COBOL

En el anexo [L] puede ver ambos programas ya codificados.

4.2.3 Programas de pruebas: Cobol - DB2

En este apartado vamos a ilustrar el desarrollo de los programas COBOL–DB2, en total hemos construido seis programas: Tres programas de consultas y tres programas de inserción.

Para este desarrollo tendremos primero que generar las tablas que propusimos en el apartado de diseño, estas tablas, y en general cualquier comando DB2 que queramos ejecutar en el mainframe lo haremos mediante menú interactivo de funciones DB2 en el mainframe, y una vez ahí, en el **SPUFI** ejecutaremos los mandatos SQL. Si desea consultar como acceder al SPUFI en el mainframe consulte la referencia [3]: DB2 en el mainframe.
En el anexo [M] podemos ver el código SQL utilizado para la creación de las tablas.

Una vez creadas las tablas con sus relaciones, hemos elaborado tres programas que cuantifican el tiempo que tarda en ejecutarse el tipo de consulta que implementan, en nuestro caso:

- Programa **SELCRUZ**: Este programa implementa un doble cruce con dos **INNER JOIN** (figura 4-13).

```
SELECT COUNT(T.EMPNO)
INTO   :COUNTR, :RETCOUNT
FROM (
    SELECT R.EMPNO
    FROM (
        SELECT P.PROJNO, P.PROJNAME, E.EMPNO
        FROM KC02520.PROJ P
        INNER JOIN
        KC02520.EMPPROJACT E
        ON P.PROJNO = E.PROJNO
        ) R
    INNER JOIN KC02520.EMP EM
    ON R.EMPNO = EM.EMPNO
    ) T
```

Figura 4-13 Medición de doble cruce.

- Programa **SELNOT:** Este programa implementa un doble NOT EXISTS (figura 4-14).

```
SELECT COUNT(CE.EMPNO)
INTO   :COUNTR, :RETCOUNT
FROM (
    SELECT B.EMPNO
    FROM KC02520.EMP B
    WHERE NOT EXISTS (
        SELECT A.EMPNO
        FROM KC02520.EMP A
        WHERE NOT EXISTS (
            SELECT EMPNO
            FROM KC02520.EMPPROJACT T
            )
        )
    ) CE
```

Figura 4-14 Consulta de medición del doble NOT EXISTS

- Programa **SELAVG:** Este programa se encarga de probar las funciones agregadas, en nuestro caso implementa la media (figura 4-15).

```
EXEC SQL


          SELECT COUNT(AV.EMPNO)
          INTO   :COUNTR, :RETCOUNT
          FROM (
              SELECT EMPNO, AVG(EMPTIME)
              FROM KC02520.EMPPROJACT
              GROUP BY EMPNO
              HAVING AVG(EMPTIME) < 5
             ) AV


END-EXEC
```

Figura 4-15 Consulta de medición de funciones agregadas

Desarrollo para cada programa

A partir de este punto vamos a detallar el desarrollo para un único programa, este proceso lo hemos seguido para los tres programas de consultas.

Una vez que identificamos las tablas que aparecen en nuestra consulta debemos de declarar en el programa lo que se conoce como DCL, una asociación entre las variables COBOL del programa y los atributos de la tabla que queremos usar en nuestro programa. Esta DCL se compone de 2 partes: la declaración de la tabla en SQL y la declaración de sus variables en COBOL.

```
************************************************************************
* DCLGEN TABLE(KC02520.EMPPROJACT)                                     *
*        LIBRARY(KC02520.DCLEMP.COBOL)                                 *
*        ACTION(REPLACE)                                               *
*        LANGUAGE(COBOL)                                               *
*        QUOTE                                                         *
* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS         *
************************************************************************
     EXEC SQL DECLARE KC02520.EMPPROJACT TABLE
     ( EMPNO                          CHAR(6) NOT NULL,
       PROJNO                         CHAR(6) NOT NULL,
       ACTNO                          SMALLINT NOT NULL,
       EMPTIME                        DECIMAL(5, 2),
       EMSTDATE                       DATE,
       EMENDATE                       DATE
     ) END-EXEC.
************************************************************************
************************************************************************
* COBOL DECLARATION FOR TABLE KC02520.EMPPROJACT                       *
************************************************************************
 01  DCLEMPPROJACT.
     10 EMPNO                PIC X(6).
     10 PROJNO               PIC X(6).
     10 ACTNO                PIC S9(4) USAGE COMP.
     10 EMPTIME              PIC S9(3)V9(2) USAGE COMP-3.
     10 EMSTDATE             PIC X(10).
     10 EMENDATE             PIC X(10).
************************************************************************
* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 6             *
************************************************************************
```

Figura 4-16 DCL de la tabla EMPPROJACT del programa SELCRUZ

Esta DCL debe de ir en la working storage section del programa COBOL, se puede implementar a mano, pero el mainframe puede generarnos esta DCL si ya tenemos la tabla creada, para ello, en el menú interactivo de funciones DB2, en el menú DCLGEN se puede especificar la información necesaria para generarla automáticamente.

También ha sido necesario incluir en esta sección del código la SQLCA o área de comunicación entre el DB2 y el programa COBOL. Una vez que hayamos incluido la información relativa a DB2, solo queda implementar la lógica de nuestro programa. Hemos reutilizado los párrafos de cálculo de tiempo implementados en el programa de pruebas BATECOB para medir los tiempos e imprimir un log de salida.

Desarrollo de programas con Inserciones

Para el desarrollo de los programas de inserción se sigue una metodología idéntica a la anterior, lo que cambia es el código SQL que ejecutaremos en nuestro párrafo Proceso. También añadiremos un bucle que nos insertará los N registros que le indiquemos en el código. Es necesario ir incrementando la clave del registro que insertemos con cada iteración, esto se hace para no violar la restricción de primary key de la tabla que estemos modificando.

```
PERFORM 900000 TIMES
EXEC SQL
     INSERT INTO KC02520.EMPPROJACT
               (EMPNO,
                PROJNO,
                ACTNO,
                EMPTIME,
                EMSTDATE,
                EMENDATE)
          VALUES(:EMPNO,
                 :PROJNO,
                 :ACTNO,
                 :EMPTIME,
                 :EMSTDATE,
                 :EMENDATE)
END-EXEC
```

```
   IF SQLCODE NOT= ZERO
        DISPLAY 'REGISTRO ERRONEO: '
        DISPLAY SQLCODE
        DISPLAY DCLEMPPROJACT
   END-IF
   ADD 1 TO PROJNO-AUX
   MOVE PROJNO-AUX TO PROJNO
   ADD 1 TO ACTNO
   END-PERFORM



   .
2000-PROCESO-EXIT.
```

Figura 4-17 Párrafo proceso del programa de inserción INSEMAC

Puede encontrar estos programas en el anexo [L].

5 Integración, pruebas y resultados

En este apartado evaluaremos los resultados de la ejecución de nuestros programas y formularemos una breve conclusión, ya que nuestros programas son programas de prueba y no forman parte de ninguna aplicación en sí misma que estemos testeando..

5.1 Programas sin DB2

En este apartado se recogen todas las pruebas y resultados de los programas que no contienen acceso a DB2. Se analizaran los resultados de las pruebas de COBOL & C y se evaluara una conclusión final.

5.1.1 Programa Apertura de ficheros Cobol & C: Resultados y conclusiones

A continuación ilustramos la salida de los dos programas (figuras 5-1 y 5-2) y evaluaremos un resultado de la prueba.

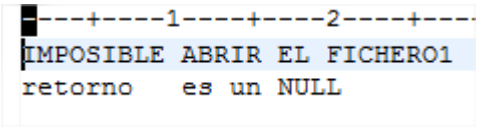


Figura 5-1 Salida del programa PROGC capturada desde RDZ

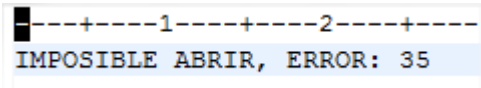


Figura 5-2 Salida del programa COBOL2 capturada desde RDZ

Tras probar estos dos sencillos programas observamos que por sí solas las funciones COBOL diferencian los tipos de errores, llegamos a la conclusión de que las funciones de COBOL son más atractivas a la hora de implementar un control de errores más detallado. Esto es una característica que a la hora de desarrollar aplicaciones de negocio simplifica muchísimo la tarea de detectar errores y corregir incidencias y recordemos: implementado todo dentro del mismo lenguaje, estandarizado y validado.

5.1.2 Programa batería de pruebas: Cobol & C

En este apartado evaluaremos la salida de los programas que median el tiempo de ejecución de las operaciones aritméticas, asignación de variables y de ficheros en lenguaje C y COBOL.

Tras realizar sucesivas pruebas, configuramos los parámetros para las variables que indicaban el número de veces que se ejecuta cada instrucción en 60.000.000 iteraciones para cada operación de fichero y 1.000.000 iteraciones para cada operación aritmética y de asignación de variables.

Figura 5-3 Configuración de parámetros para la prueba

Después de ejecutar ambos programas obtuvimos las siguientes mediciones de tiempos de ejecución para las operaciones:

OPERACION	SEGUNDOS	CENTÉSIMAS
Escritura	12	30
Lectura	4	55
Suma	0	02
Resta	0	02
Multiplicación	0	02
División	0	03
Potencia	0	03
Swap	0	03

Tabla 5-1 Resultados obtenidos por el programa de medición de operaciones C (BATEC)

OPERACION	SEGUNDOS	CENTÉSIMAS
Escritura	02	93
Lectura	02	91
Suma	00	08
Resta	00	07
Multiplicación	00	09
División	00	14
Potencia	01	87
Swap	00	01

Tabla 5-2 Resultados obtenidos por el programa de medición de operaciones COBOL (BATECOB)

En el anexo [N] puede encontrar la salida de los programas.

Empezaremos analizando las operaciones aritméticas, observamos que la diferencia entre la suma y la resta en C es más rápida que en COBOL, también lo son la multiplicación y la división, esto se debe a que COBOL no es fuerte a la hora de ejecutar operaciones aritméticas y aquí lo estamos comprobado, la diferencia de tiempo oscila en cuatro veces más rápida la ejecución aritmética en C que en COBOL.

Si observamos la operación de swap la cosa ya va cambiando a favor de COBOL y en contra de C, observamos que Swap es alrededor de tres veces más rápido en la ejecución de COBOL que en C, esto se debe a que ya nos estamos adentrando en el terreno de COBOL, el manejo de los datos.

Por último, si observamos los tiempos de las operaciones de ficheros tenemos un claro vencedor: COBOL, en operaciones de lectura aventaja a C siendo caso dos veces más rápido y en operaciones de escritura aventaja a C siendo cuatro veces más rápido. Y todo esto recordemos que implementando en su interior control de errores adicional a C.

Con esto llegamos a la conclusión de que, para el manejo de los datos, a nivel de operación y inclusive teniendo más funcionalidad en sus funciones, COBOL es más rápido que un lenguaje con un compilador tan ligero como lo es C. Por esto COBOL es un lenguaje muy a tener en cuenta a la hora de implementar aplicaciones de negocio que precisen grandes volúmenes de datos.

5.2 Programas de prueba con DB2

En este apartado se recogen los resultados de las pruebas de los programas de mediciones de tiempo DB2, no se evaluarán las pruebas de los programas de insertar ya que no es el propósito de esta prueba.

5.2.1 Programas de pruebas: Cobol - DB2

En este apartado vamos a evaluar los resultados de las mediciones de tiempo de los programas que ejecutaban diferentes tipos de consultas. Hemos hecho las mediciones para tres tamaños de tablas diferentes: 10.000, 100.000 y 900.000 registros respectivamente.

Funciones Agregadas: Este programa ejecuta una consulta sobre la tabla EMPPROJACT que relaciona los empleados con los proyectos en los que trabajan, hemos hecho la media por empleado del tiempo imputado a cada proyecto. Hemos fijado el mismo número de empleado para que se haga la media sobre el mismo empleado y así forzar a la función a ejecutar una media de N registros que contenga la tabla.

Nº REGISTROS	SEGUNDOS	CENTÉSIMAS
10.000	00	04
100.000	00	21
900.000	01	06

Tabla 5-3 Resultados obtenidos de las ejecuciones del programa COBOL (SELAVG) para diferentes tamaños de las tablas

Observamos que el crecimiento no llega a ser lineal, si multiplicamos por 10 el número de registros el crecimiento en tiempo todavía no ha llegado ni a ser lineal en los tres tamaños de tablas.

Doble NOT EXISTS: Esta consulta devuelve los empleados no existan en la tabla que relaciona con los proyectos, luego al volver a hacer la consulta, como la primera consulta es nula, la segunda será la tabla completa.

Nº REGISTROS	SEGUNDOS	CENTÉSIMAS
10.000	00	06
100.000	00	59
900.000	04	52

Tabla 5-4 Resultados obtenidos de las ejecuciones del programa COBOL (SELNOT) para diferentes tamaños de las tablas

Observamos que la diferencia entre 10.000 y 100.000 es mínima. Para la magnitud del número de registros, el crecimiento no llega ni a ser lineal aunque esta próximo. De todas maneras, son tiempos muy rápidos para consultas tan pesadas.

Doble cruce: Al cruzar las tres tablas a la vez conseguimos triplicar la información, con tantos registros comprobaremos como cruza.

Nº REGISTROS	SEGUNDOS	CENTÉSIMAS
10.000	00	08
100.000	00	67
900.000	05	79

Tabla 5-5 Resultados obtenidos de las ejecuciones del programa COBOL (SELCRUZ) para diferentes tamaños de las tablas

En el cruce observamos que, no llega por muy poco pero ya se empieza a apreciar que está llegando a ser lineal la duración respecto al incremento de número de registros. Aun así creemos que estas mediciones para la magnitud del número de los registros son muy buenas para los tres casos. En el anexo [O] podemos ver la salida de estos programas.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

A lo largo de este estudio hemos ilustrado las características de todo un entorno real de desarrollo de aplicaciones de negocio. Hemos evaluado sus tecnologías y las hemos puesto a prueba con resultados más que satisfactorios a pesar de que estas tecnologías sean algunas realmente antiguas.

También hemos ilustrado cómo el z/OS es capaz de lograr las metas tan exigentes que se le presentan en los diferentes entornos en los que se implanta. Hemos visto que el sistema hardware que lo compone es muy potente, se puede escalar y a su vez es muy fiable a la hora de confiar en él cualquier negocio.

En relación al z/OS observamos que es un sistema operativo bastante complejo, lleno de componentes que se encargan de cumplir tareas específicas que, en conjunto, dan lugar a impresionantes resultados. Pero esta complejidad es necesaria en comparación con todo lo que puede ofrecer. En definitiva podría decirse que es muy robusto, capaz de gestionar a múltiples usuarios conectados él concurrentemente, hemos comprobado que es capaz de soportarlo a la perfección.

En relación al desarrollo de programas, hemos visto que es posible desarrollar en el mainframe con un IDE basado en eclipse, el RDZ, o directamente desde el TSO. Es un entorno de desarrollo perfectamente preparado para la programación y desarrollo de aplicaciones.

Tanto teórica como prácticamente hemos ilustrado gran parte del entorno mainframe de IBM, pero también hemos ilustrado que, aunque COBOL tiene gran antigüedad entre los lenguajes de programación en activo, la verdad es que sigue en el mercado por derecho propio. Tras ponerlo a prueba en multitud de ocasiones en este estudio hemos llegado a la conclusión de que es un lenguaje fiable, seguro y muy potente para el tratamiento de datos. Ha recortado en tiempos de ejecución a lenguajes con compiladores muchísimo más ligeros como el lenguaje C.

Hemos hecho pruebas de operaciones costosas al gestor de BBDD del mainframe, el DB2, y nos ha sorprendido gratamente la capacidad que tiene de procesar cantidades considerables de datos. Ejecutando consultas realmente pesadas en tablas con tamaños de registros realmente grandes hemos comprobado que sus tiempos de ejecución son verdaderamente rápidos.

En definitiva, creemos que cuando las grandes empresas usan este entorno en su negocio, lo hacen con fundamento. Este sistema posee unas garantías que pocos sistemas del mundo pueden superar.

6.2 Trabajo futuro

Como trabajo futuro se propondría seguir investigando esta tecnología y seguir profundizando cada vez más, opinamos que es muy potente.

En este estudio hemos visto solo una visión general de lo que entornos de este tipo ofrecen. Estas características hacen que las grandes empresas necesiten este tipo de entornos.

Como mejora se podría proponer investigar más la parte online, probar el CICS y ejecutar transacciones, algo de lo que este sistema está especializado gracias al CICS, el gestor transaccional de los mainframe de IBM.

Podrían ilustrarse el funcionamiento de una transacción que hiciera la llamada a varias pantallas y estas hagan llamadas a rutinas para implementar una aplicación COBOL–DB2 online.

Sería interesante estudiar este campo, ya que en este estudio nos hemos centrado en dar una visión más global y de presentación del entorno, de la parte batch y de desarrollo de programas.

También sería interesante proponer el desarrollo de una aplicación batch de transformación de datos ilustrando lo que sería un data warehouse o un data mart, aplicación donde COBOL es más que recomendable. Sería interesante planificar una cadena de JCL que se encargaran de la transformación de datos periódicamente con una planificación en tiempo real como se hace en la actualidad, añadiéndole periodicidad diaria o semanal e ilustrar su funcionamiento.

Referencias

[1] Información de la preparación de un programa con acceso a DB2 : http://www-01.ibm.com/support/knowledgecenter/SSEPEK_10.0.0/com.ibm.db2z10.doc.intro/src/tpc/db2z_programprepprocesses.dita?lang=es

[2] Alocación de ficheros en JCL: http://www.adictosaltrabajo.com/tutoriales/jcl-intro/

[3] DB2 en el mainframe: https://www-01.ibm.com/support/knowledgecenter/SSEPEK_10.0.0/com.ibm.db2z10.doc.intro/src/tpc/db2z_tut_querydata.html.

[4] Documentación ISPF: https://www-01.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/com.ibm.zos.v2r1.f54ug00/toc.htm

[5] Documentación sobre RDZ: http://www-01.ibm.com/support/knowledgecenter/SSQ2R2_9.1.1/com.ibm.wsentdev.doc/topics/c_node_configuring.html?lang=es

[6] Información COBOL I: http://www.escobol.com/

[7] Información COBOL II: http://www.consultoriocobol.com/

[8] Brian W. Kernighan, Dennis M. Ritchie, El lenguaje de programación C (2 ªED, Pearson educación).

[9] Rick Long, Mark Harrington, Robert Hain, Geoff Nicholls, IMS Primer (www.redbooks.ibm.com).

[10] Karan Singh, Paul Rogers, ABCs of IBM z/OS System Programming Volume 1 (Fourth Edition (August 2014)), (www.redbooks.ibm.com).

[11] Bart Steegmans, Ulrich Gehlert, Judy Ruby-Brown, Paul Tainsh, DB2 for z/OS and OS/390: Ready for Java, (www.redbooks.ibm.com).

[12] Paolo Bruni Neale Armstrong Ravi Kumar Kirsten Ann Larsen Tink Tysor Hao Zhang, Extremely pureXML in DB2 10 for z/OS (www.redbooks.ibm.com).

[13] Babette Haeusser Luciano Cecchetti Rafael Garcia Lecuona Daniel Rincon Lopez, IBM Virtualization Engine TS7700 Release 1.4a: Tape Virtualization for System z Servers (www.redbooks.ibm.com).

[14] IBM Redbooks Solution Guide, Automated Conversions to IBM DB2 for z/OS (www.redbooks.ibm.com).

[15] IBM Redbooks Solution Guide, Optimizing Database Administration with IBM DB2 Autonomics for z/OS (www.redbooks.ibm.com).

[16] IBM Redbooks Solution Guide, Securing the IBM Mainframe (www.redbooks.ibm.com).

[17] James O'Grady Ian Burnett Jim Harrison San Yong Liu Xue Yong Zhang, Application Development for IBM CICS Web Services (www.redbooks.ibm.com).

[18] Tim Rowe, Tools and Solutions for Modernizing Your IBM i Applications (www.redbooks.ibm.com).

[19] Chris Rayns Jeremy Hamilton Peter Henningsen Kenichi Yoshimura, IBM Application Development and Problem Determination (www.redbooks.ibm.com).

[20] Paolo Bruni Zhen Hua Dong Josef Klitsch Maggie Lin Rajesh Ramachandran Bart Steegmans Andreas Thiele, DB2 for z/OS and WebSphere Integration for Enterprise Java Applications (www.redbooks.ibm.com).

[21] Reginaldo Barosa Surya Kumar Bose Scott Davis Jeremy Hamilton Joachim Kaltenbach David Lawrence Veronique Quiblier Zhenhua (Eric) Shen Marc Terwagne Kenichi Yoshimura, Topics on Version 7 of IBM Rational Developer for System z and IBM WebSphere Developer for System z (www.redbooks.ibm.com).

Glosario

DB2	Data Base 2
RDZ	Rational Developer for system Z

JCL	Job Control Language
COBOL	COmmon Business-Oriented Language
SQL	Structured Query Language
BBDD	Bases de Datos
RAS	Reliability, Availability, Serviceability
TSO	Time Sharing Option
ISPF	Interactive System Productivity Facility
IBM	International Business Machines
DBRM	Data Base Request Module
SQLCA	SQL Communication Area
PSW	Program Status Word
LPAR	Logical Partition
WLM	Workload Management
CICS	Customer Information Control System
E/S	Entrada/Salida
DAT	Direct Address Translate
BIND	Proceso por el cual un programa debe pasar para poder ejecutarse correctamente su información DB2 en un mainframe.
PU	Processing Unit
SMP	Symetric Multi Processing
Precompilador DB2	Elemento que separa la información DB2 del programa y la sustituye por llamadas a funciones compilables por el compilador.
Compilador	Programa que se encarga de transformer un programa escrito en un lenguaje de programación cualquiera a código máquina.
Enlazado	Proceso por el cual un programa en código objeto se transforma para construir un programa ejecutable.

Anexos

A El desarrollo del Parallel Sysplex

A principios de los años 1990 el hardware instalado predominante para el mainframe era el ES / 9000. Este sistema era capaz de procesar 500 millones de instrucciones por segundo (MIPS).

Sin embargo, esta tecnología requería una gran cantidad de consumo energético y generaba gran cantidad de calor por lo que requerían un sistema de refrigeración suplementario que incluía un circuito de agua helada.

La tecnología de bajo costo fue ganando popularidad en el mundo de los negocios, así como la relativamente nueva tecnología en procesadores RISC que ejecutan UNIX.

Todos estos sistemas estaban basados en la tecnología de semiconductor de óxido metálico complementario (CMOS). En 1992 IBM decidió utilizar a la tecnología CMOS en sus servidores de mainframe.

Este cambio en los procesadores presentaba algunos problemas. El principal era que la tecnología CMOS era mucho más lenta que la de los procesadores bipolares. El procesador CMOS más rápido en ese momento procesaba a 60 MIPS, lo cual era menos de 12% de la velocidad de los procesadores bipolares más utilizados.

La solución consistió en ejecutar la carga de conjuntamente entre varios procesadores en paralelo, aumentando así el rendimiento del sistema. Este sistema es lo que se conoce como Complex System Parallel o de forma abreviada Parallel Sysplex.

B El Hardware que el Z/OS controla

En la imagen, se puede ver todo el hardware que el z/OS controla:

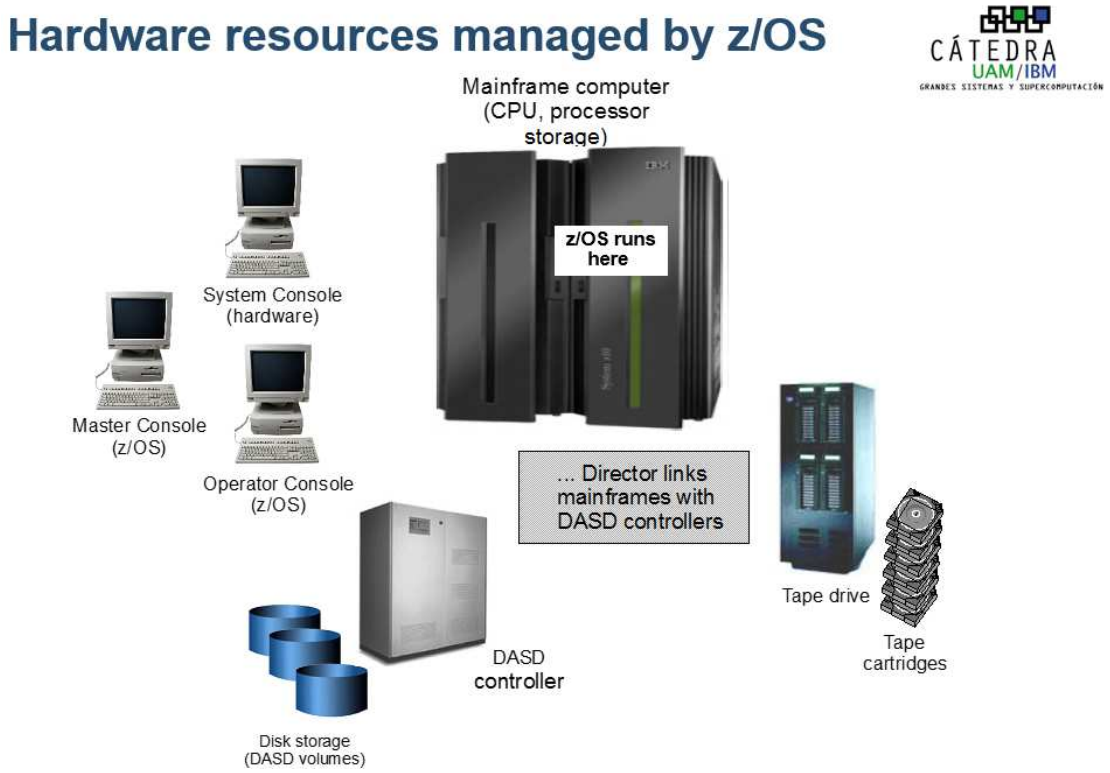


Figura 0-1: Algunos periféricos del z/OS.

En la imagen se puede ver tanto los periféricos que es capaz de controlar como cintas, discos DASD y el controlador DASD. El controlador DASD es un dispositivo capaz de controlar los volúmenes DASD.

Un volumen DASD se utiliza para guardar datos y programas ejecutables. Estos volúmenes se identifican mediante sus etiquetas DASD y el espacio en el que se encuentran pueden ser reasignados para reutilizarse. En un volumen DASD el nombre de un conjunto de datos debe ser único. Por último comentamos que la estructura de archivos del z/OS no es jerárquica como la estructura de los archivos UNIX: Un conjunto de datos no tiene un equivalente a un nombre de ruta.

También se muestran los tres tipos de consolas que permiten acceder al sistema: la consola del sistema (system console), la consola maestra (master console) y la consola del operador (operator console).

C El almacenamiento virtual del Z/OS

A grandes rasgos, el almacenamiento virtual no es otra cosa que disponer de un espacio de direcciones mayor que el que en la realidad la memoria puede ofrecer. Esto significa que unas cuantas direcciones de memoria de nuestro espacio de direcciones no van a existir en memoria real.

El z/Os implementa esta virtualización mediante lo que se conoce como paginación. Esto consiste en dividir la memoria en páginas de información y guardar el número máximo posible que quepan en memoria principal (las elegidas suelen ser las más usadas pero esto depende de la política de gestión que se use). El resto de páginas se guardan en almacenamientos secundarios o auxiliares, con la consiguiente lentitud en el acceso (figura 0-2).

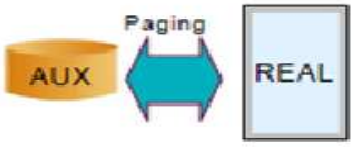


Figura 0-2: La forma del z/OS de gestionar la memoria es la Paginación.

En z/Os los bloques de datos son del mismo tamaño (4 Kilobytes) y según su localización se denominan de diferente manera (figura 0-3)):

- Un bloque de almacenamiento real es un frame.
- Un bloque de almacenamiento virtual es una página.
- Un bloque de almacenamiento auxiliar es un slot.

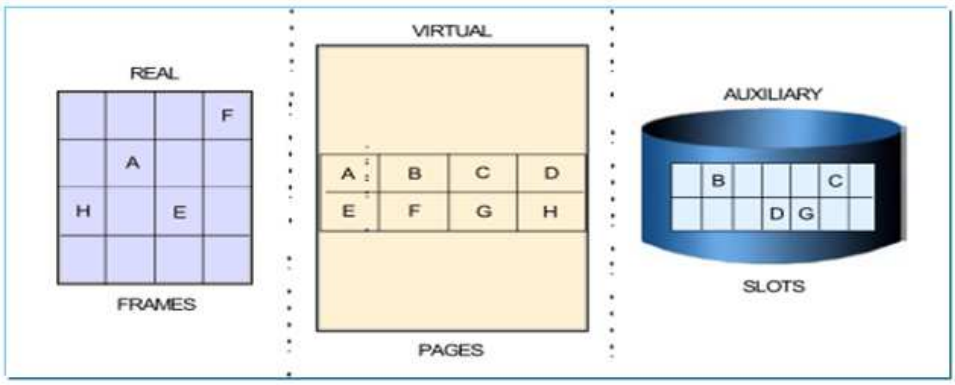


Figura 0-3: Páginas, frames y slots

El almacenamiento virtual se divide en segmentos de 1 megabyte compuestos de páginas de 4 kilobytes. La transferencia de bloques de datos entre el almacenamiento auxiliar y almacenamiento real se llama paginación, esto se hace cuando unos datos solicitados no se encuentran en almacenamiento real (no se encuentra el frame), entonces se produce un evento (fallo de memoria) que señaliza al sistema sobre este incidente para que traiga la los datos de la memoria auxiliar (slot) al almacenamiento real.

Para gestionar este sistema de intercambio de paginas entre memoria real y auxiliar el z/OS utiliza segmentos, páginas y regiones de tablas para mantener el seguimiento del almacenamiento de toda la información, es decir, el estado y localización de cada parte de la memoria en cada instante. También tiene agentes que gestiona el estado de cada tipo de memoria en cada caso:

- Real storage manager (o RSM): Rastrea los contenidos del almacenamiento central o principal.
- Auxiliary storage manager auxiliar (o ASM): Utiliza los ficheros de páginas del sistema para mantener un seguimiento de los slots almacenados en memoria auxiliar.
- Virtual storage manager (o VSM): Responde a las solicitudes para obtener y liberar de almacenamiento virtual.

Las direcciones se traducen dinámicamente mediante Traducción Dinámica de Direcciones (DAT). Este es el proceso por el cual se traducen las direcciones de memoria virtuales a direcciones de memoria reales o físicas.

D Funcionamiento del Workload Management

El WLM es capaz de supervisar el uso de recursos de todo el sistema de recursos para determinar si se utilizan plenamente. También gestiona la E/S, es decir, selecciona los dispositivos que se asignan en cada caso (si la selección de dispositivos existe) con el fin de equilibrar el uso de dispositivos de E/S.

Determina qué espacios de direcciones deben extraerse (y cuándo). También inhibe la creación de nuevos espacios de direcciones o roba páginas cuando existen ciertas carencias de almacenamiento en la memoria principal o central y es capaz de cambiar la prioridad de envío de espacios de direcciones. De esta manera WLM es capaz de controlar la velocidad a la que se permite que los espacios de direcciones consuman recursos del sistema.

Pero no solo eso, otros componentes de z/OS como los gestores de transacciones (CICS) y gestores de bases de datos (DB2) se pueden comunicar con el WLM para indicar un cambio de estado en un espacio de direcciones en particular (o en el sistema en su conjunto) con el fin de invocar el poder de toma de decisiones del WLM.

Por ejemplo, el WLM es notificado cuando ocurren los siguientes eventos:

- El almacenamiento central está configurado dentro o fuera del sistema.
- Se va a crear un nuevo espacio de direcciones.
- Se va a eliminar un espacio de direcciones.
- Un intercambio entre jerarquías diferentes de memoria comienza o termina (Gestión de memoria virtual).
- Las rutinas de asignación tienen que elegir los dispositivos que se destinarán a una solicitud.
- Una carga de trabajo de prioridad está faltando a su objetivo.

En definitiva, el Workload Management es una pieza fundamental que se encarga de asegurar el buen rendimiento de todas las operaciones que se ejecutan en el z/OS, siendo el encargado de la toma de decisiones para asegurar la consecución de los objetivos para los que se configuró el sistema.

E Tratamiento de interrupciones en el z/OS

El z/OS usa 6 tipos de interrupciones:

-**Supervisor calls o SVC interrupts**: Se interrumpe el programa que se está ejecutando y se pasa el control al supervisor, de tal forma que se ejecuta el servicio que el programa solicitó a partir de la cual se generó esta interrupción. Se suelen lanzar cuando se requieren de ciertos servicios como: abrir un fichero, obtener memoria o cuando se manda un mensaje al operador del sistema. Estos servicios se solicitan mediante macros como OPEN (ficheros), GETMAIN (memoria) o WTO (mensaje al operador).

-**I/O interrupts**: Se producen cuando se completa una operación de E/S, por ejemplo, cuando un dispositivo como una impresora está listo para su uso o cuando ocurre algún tipo de error en la operación E/S.

-**External Interruptions**: Indican eventos externos, como el final de un intervalo de tiempo, un procesador recibiendo la señal de otro procesador, o cuando el operador presiona la tecla de interrupción en la consola.

-**Restart Interrupts**: Ocurre cuando el operador selecciona la función reinicio desde la consola o cuando una instrucción de reinicio SIGP de otro procesador es recibida.

-**Program Interrupts**: Son lanzadas desde los programas, por ejemplo, cuando un programa intenta ejecutar una acción inválida o cuando se produce un fallo de pagina.

-**Machine check interrupts**: Son provocadas por un mal funcionamiento de la maquina.

Un concepto clave para el sistema de tratamiento de interrupciones es la PSW (Program Status Word). Junto a la PSW también se encuentran los siguientes registros que guardan información actual de la ejecución del programa:

- **Access Registers**: Que especifican el espacio de direcciones donde se encuentran los datos.
- **General Registers**: Que almacenan datos de usuarios y direccionan los datos almacenados.

La PSW del programa contiene información sobre el programa que se está ejecutando actualmente. Incluye la dirección de la siguiente instrucción del programa y diversa información de control del mismo.

Para ilustrar el funcionamiento de la PSW diremos que existen tres tipos de PSW: La PSW actual (que indica la siguiente instrucción a ejecutar), la nueva PSW (que contiene la rutina que puede procesar la interrupción asociada) y la PSW antigua que sirve como almacenamiento temporal en caso de la llegada de una interrupción.

El proceso de interrupción cuando el procesador está habilitado para dicha instrucción funciona de la siguiente manera: Cuando llega una instrucción y el procesador está habilitado para procesarla, se intercambian las PSW. La PSW actual se almacena en la PSW antigua registrando el tiempo en el que se produjo la interrupción para después, cargar el contenido de la nueva PSW en la PSW actual.

F Locking – Evitando interbloqueos

Un candado o **lock** es un campo que indica si un recurso está siendo usado y por quién.

Existen dos tipos de candado en z/OS: globales y locales. Los globales son para recursos con más de un espacio de direcciones mientras que los locales son para recursos asignados a un solo espacio de direcciones.

Para usar un recurso protegido con un candado, una rutina debe solicitar el candado del recurso. Si el recurso no está disponible, el agente solicitante actuará de una manera u otra según sea el candado: ***spin lock*** o ***suspend lock***.

- ***Spin lock***: Este tipo de candado funciona de manera que el solicitante continúe probando el candado hasta que el recurso esté disponible. Cuando se libere, el solicitante podrá acceder a él y a su recurso. Este tipo de candados suelen ser globales.
- ***Suspend lock***: Si el recurso no está disponible, el solicitante se retrasa hasta que el recurso esté disponible. Mientras, otro trabajo es atendido en el procesador hasta que el candado se libere. Suelen tener este tipo de características los candados locales.

Una vez explicado los tipos de candados de los que dispone z/OS, vamos a explicar cómo los utiliza para evitar los deadlocks.

En z/OS, los candados son organizados jerárquicamente, de tal modo que un procesador o rutina solo puede solicitar candados de una jerarquía mayor a los candados que actualmente tiene.

En el ejemplo que proponíamos anteriormente, el proceso PA cuenta con el candado A, pero necesita el B. Del mismo modo, el proceso PB cuenta con el candado B y necesita el candado A.

Este ejemplo anterior no se podría dar, dado que los candados deben adquirirse en una secuencia jerárquica. El candado A precede al candado B en la jerarquía por lo que, en z/OS PB no puede solicitar el candado A hasta que se libere el candado B. Para ello, debe de liberar B, solicitar A y una vez conseguido A, solicitar el B.

G Programa SORT

Un “paso” SORT en un JCL se encarga de ordenar los registros que se le pasan por la entrada y formatear la salida a un fichero.

```
//SORT001    EXEC   PGM=SORT,PARM=('DYNALLOC=(SYSALLDA,32)')
//SORTIN        DD      DSN=nombre.fichero.entrada1,DISP=SHR
//              DD      DSN=nombre.fichero.entrada2,DISP=SHR
//SORTOUT            DD          DSN=nombre.fichero.salida1,
//                DISP=(,CATLG,DELETE),SPACE=(CYL,(500,100))
//SYSOUT                        DD                    SYSOUT=*
//SYSPRINT                      DD                    SYSOUT=*
//SYSIN                            DD                      *
 SORT FIELDS=(I,L,T,O,I,L,T,O)
```

Esta es la estructura general de un paso SORT en un JCL, a continuación vamos a explicarlo en detalle:

`//SORT001  EXEC PGM=SORT,PARM=('DYNALLOC=(SYSALLDA,32)') :` En esta línea se indica el nombre del paso del programa (***`SORT001`***) y el programa a ejecutar, en este caso el programa SORT (*`PGM=SORT`*). También se le indica la cantidad de memoria que se usara para este paso (***`PARM=('DYNALLOC=(SYSALLDA,32)')`***) y puede tomar valores de múltiplos cuadráticos de 8.

`//SORTIN  DD  DSN=nombre.fichero.entrada1,DISP=SHR` : La **SORTIN** nos indica el nombre de los ficheros de entrada y si existen o no, esto se indica con **`DISP=SHR`** si existe o con **`DISP=NEW`** en el caso de que no existiera, aunque no se utiliza en las entradas de SORT por razones obvias.

A continuación se indica el fichero de salida especificando su nombre y sus características de almacenamiento en la maquina. También se indican las salidas del sistema y se especifica la SYSIN, en la que se indica el tipo de SORT que se va a realizar aunque en nuestro caso vamos a ilustrar el más común.

`SORT FIELDS=(I,L,T,O,I,L,T,O):` Esta línea especifica el tipo de SORT que vamos a realizar, en nuestro caso es una ordenación de dos ficheros que se cruzarán en uno solo de salida ordenado por la clave que se especifica entre los paréntesis y que detallamos a continuación.

`I (Inicio):` Indica la posición de inicio de nuestro campo clave por el que se va a ejecutar la ordenación.
`L (Longitud)`: Longitud máxima del campo clave por el que vamos a ejecutar la ordenación.
`T (Tipo de dato):` Tipo de dato de la clave por el que se va a efectuar la ordenación:

- CH: Indica que el campo es de tipo alfanumérico.

- PD: Indica que el campo es de tipo numérico empaquetado.
- FI: Indica que el campo es de tipo hexadecimal con signo.
- BI: Indica que el campo es de tipo hexadecimal sin signo.
- ZD: Indica que el campo es de tipo numérico sin comprimir.

`O (Orden de la ordenación):` Puede ser una ordenación ascendente `(A)` o descendente `(D).`

Este patrón puede repetirse para tantas claves como se desee ejecutar la ordenación.

H Tabla de códigos de retorno de las funciones de tratamiento de ficheros COBOL.

Código	Descripción
00	La operación fue exitosa.
02	La sentencia de INPUT o OUTPUT fue ejecutada exitosamente, pero se detectó una clave duplicada. Este tipo de error aplica a archivos indexados.
04	La sentencia de lectura fue ejecutada exitosamente, pero la longitud del registro leído no coincide con los atributos especificados para ese archivo.
05	La sentencia de open fue ejecutada exitosamente, pero el archivo opcional referenciado no está presente en el momento en que el OPEN se ejecuta.
10	Se alcanzó el fin de archivo EOF cuando se ejecutó una operación de lectura.
35	El fichero no existe.
47	Se intenta leer un archivo que no se abrió en modo INPUT o I-O.
99	Archivo bloqueado por un usuario

I Estructura de un programa COBOL.

En este anexo vamos a explicar la estructura de un programa COBOL cualquiera, cómo está estructurado y algunas de sus directivas más interesantes.

Como cualquier lenguaje de programación, COBOL tiene sus verbos, sus *frases* y sus *párrafos.* COBOL proviene del inglés, fue creado de tal manera que fuese lo mas similar posible a éste a la hora de expresar lo que queremos hacer en el programa.

COBOL es un lenguaje estructurado, está formado por varias **divisiones** que tienen su propia función dentro del programa. Además, COBOL es muy ordenado a la hora de su escritura, *'todo tiene que ir en su sitio'*, un programa COBOL está formado por sentencias o *frases* de 80 caracteres o menos. Las cuales se distribuyen de la siguiente manera en la figura 0-4 y que describiremos a continuación:

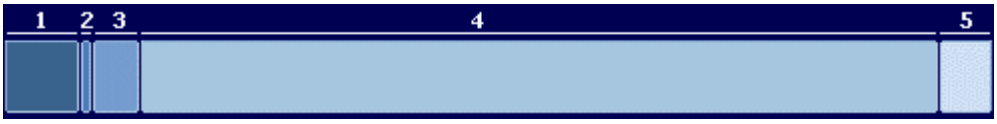


Figura 0-4: estructura de una línea de código COBOL.

- **Región 1**: Esta región ocupa los primeros 6 caracteres de la línea (figura 0-4, columnas de la 1 a la 6). Sirve para numerar las líneas de código aunque actualmente se encuentra casi en desuso y prácticamente no se utilizan.

- **Región 2**: Esta región nos ocupa un carácter (figura 0-4, columna 7) que puede ser bien un guión '-' que nos indicaría que la línea es continuación de la anterior que por su tamaño no cabía en una sola línea o bien un asterisco '*', que nos indicaría que lo que viene a continuación es una línea comentada. El compilador debe bien ignorarla o bien interpretarla como alguna opción del compilador. Esto depende del compilador que estemos usando.

- **Región 3**: A esta región también se le denomina "área A" y nos ocupa 4 caracteres (figura 0-4, columnas de la 8 a la 11). Es aquí donde se escriben los nombres de las divisiones, el nombre de los párrafos, los indicadores de FD (File Description) y los niveles de las variables 01 y 77 que posteriormente explicaremos.

- **Región 4**: También llamada “área B”, ocupa 60 caracteres (figura 0-4, columnas de la 12 a la 72). Aquí es donde se alojaran las instrucciones del programa COBOL y los índices de variables superiores a 01 y las declaraciones de las secciones del programa.

- **Región 5**: Esta región nos ocupa 7 caracteres (figura 0-4, columnas de la 73 a la 80), no se utiliza y por tanto actualmente es ignorada por el compilador.

Las líneas de código COBOL terminan con un punto ‘.’. Si el compilador no encuentra dicho punto, el compilador interpreta que la línea continúa hasta que lo encuentre. También COBOL cuenta con palabras reservadas que no debemos usar como nombres de variables o nombres de *párrafos*. Los nombres de los *párrafos* no deben de exceder los 30 caracteres de longitud.

Una particularidad del lenguaje COBOL es el tipo de datos de las variables de los que dispone, a saber: Alfanuméricos y numéricos. Las variables numéricas, al contrario que en la mayoría de los lenguajes de programación, no se miden en los bytes que ocupa el numérico en sí. En COBOL una variable de tipo numérico se mide por numero de dígitos que lo forman, es decir, si tenemos una variable que ocupa 7 dígitos, esta variable nos podrá guardar números de 0 a 9999999 si estamos hablando de valor absoluto. Si llevara signo, también los negativos serían incluidos. En el caso de las alfanuméricas el tamaño viene determinado por la forma habitual, por el número de caracteres que ocupa.

Antes de explicar qué son las *divisiones* en las que un programa COBOL se estructura, mencionaremos que existen unas variables reservadas por el propio lenguaje que tienen valores asociados especiales: HIGH-VALUES, LOW-VALUES, ZEROES o SPACES. El significado de lo que contienen es trivial. Como su nombre indica: HIGH-VALUES valores máximos, LOW-VALUES valores mínimos, ZEROES son ceros en tipo alfanumérico y SPACES son efectivamente, espacios.

Estructuración de un programa Cobol (divisiones)

Como dijimos anteriormente, un programa Cobol se organiza por *divisiones*, estas a su vez están organizadas en *secciones*, una *sección* es como un apartado dentro de una *división*.

Cualquier programa COBOL se organiza en estas 4 divisiones:

- **Identification division.**
- **Enviroment division.**
- **Data division.**
- **Procedure division.**

Identification division

Esta es la primera línea de todo programa COBOL (excluyendo comentarios). En esta división se guarda la información en relación al desarrollo del programa en el que estamos trabajando como autor, fecha de creación, maquina donde está instalado…

Esta división es meramente informativa y tiene gran utilidad para la documentación del programa:

IDENTIFICATION DIVISION.

PROGRAM-ID "*Nombre del programa*".

AUTHOR "*Nombre del autor*".

INSTALLATION "*Lugar donde está instalado (Maquina)*".

DATE-WRITTEN "*Fecha de creación*".

DATE-COMPILED "*Fecha de compilación*".

REMARKS "*Comentarios*".

Enviroment división (figura 0-5)
Esta es la segunda división que nos encontramos en nuestro programa COBOL. Esta división nos indica datos acerca del entorno del programa: el ordenador donde se ejecuta y donde se escribió este código. También contiene información acerca de los ficheros que utiliza (en el caso de que los hubiera) y cómo el programa se comunica con ellos.

Cabe destacar que, en un principio, muchas de las secciones y divisiones eran obligatorias. Hoy en día muchas no lo son, esta división perfectamente se podría omitir si para nuestro programa no tuviera ninguna utilidad. Vamos a ver más de cerca esta división y las secciones que la forman: la *configuration section* y la *input-output section*.

Configuration section
Esta sección contiene la información de los ordenadores relacionados con la creación de programa así como de su ejecución o inclusive su compilador. También contiene información para el compilador acerca de valores constantes que se definirán de aquí hasta el final del programa (SPECIAL-NAMES).
La clausula SPECIAL-NAMES se usa normalmente para cambiar la coma (,) por el punto (.) como indicador decimal en vez de la notación para así poder reservar el punto para los 'miles'.

Input-Output section
Esta última sección de la Enviroment. Se divide en dos párrafos, el párrafo FILE-CONTROL y el párrafo I-O-CONTROL.

- **FILE-CONTROL**: Se encuentra la información relativa a los ficheros declarados en el programa: como se acceden, de que tipo son, etc.
- **I-O-CONTROL**: Se encuentra información relativa a la entrada y salida del programa, no es muy usada.

```
000010 ENVIRONMENT DIVISION.
000011 CONFIGURATION SECTION.
000012 SOURCE-COMPUTER.         MAINFR.
000013 OBJECT-COMPUTER.         MAINFR.
000014 SPECIAL-NAMES.
000015
000016          DECIMAL-POINT IS COMMA.
000017
000018 INPUT-OUTPUT SECTION.
000019 FILE-CONTROL.
000020
000021      SELECT FILEPRU ASSIGN TO SYSUT1
000022         ACCESS MODE IS SEQUENTIAL
000023         FILE STATUS IS WK-STATUS-FILEPRU.
000024
```

Figura 0-5: ejemplo de Enviroment division.

La clausula **FILE-STATUS** contiene la variable que declararemos para albergar el estado de retorno de nuestro fichero, es sumamente importante y estará definida en la **Working-Storage section**, que veremos a continuación en la siguiente división.

Data división (figura 0-6)
En esta división se encuentra la definición de todas las variables, constantes y en definitiva todos los datos que utilicemos en nuestro programa. Esta división se divide en varias secciones que contienen definiciones de datos orientadas a diferentes campos, a saber:

- **File Section**
- **Working-Storage Section**
- **Linkage Section**
- **Communication Section**
- **Screen Section**

File Section

Esta sección describe los registros de todos los ficheros que vamos a usar, los que previamente habíamos definido en la enviroment division, dentro de la Input-Output section.

Este es un ejemplo de la sintaxis que seguiría la declaración del registro de un fichero:

FD Nombre del fichero.
BLOCK CONTAINS Número de registros por bloque **RECORDS**
RECORD CONTAINS Número de caracteres por registro **CHARACTERS**
LABEL RECORD Etiqueta de registro
DATA RECORD Nombre del registro.

```
000026 DATA DIVISION.
000027
000028 FILE SECTION.
000029
000030 FD FILEPRU
000031     BLOCK CONTAINS 0 RECORDS
000032     LABEL RECORD ARE STANDARD
000033     RECORDING MODE IS F
000034     RECORD CONTAINS 2 CHARACTERS
000035     DATA RECORD IS REG.
000036 01  REG                              PIC X(2).
000037
```

Figura 0-6: Ejemplo de File Section.

Working-Storage Section (figura 0-7)
Esta sección es sumamente importante ya que se utiliza para declarar las variables que usaremos en nuestro programa. Es similar a la Linkage section, solo que la Linkage section define variables que se usan para comunicarse con otros programas a los que se llama desde este programa (rutinas), esto lo veremos a continuación.

```
=COLS> ----+----1----+----2----+----3----+---
000015          Data Division.
000016          Working-Storage Section.
000017          01  DOMICILIO.
000018              02  TIPO PIC XX.
000019              02  NOMBRE PIC X(20).
000020              02  NUMERO PIC 9(4).
000021
```

Figura 0-7: ejemplo simple de la declaración de una variable en Cobol

Ahora vamos a explicar un poco más en detalle los niveles de las variables. El nivel 01 indica que es el primer nivel, aplicado al ejemplo de la figura 0-7, domicilio no pertenece a ningún nivel, mientras que nombre, número y tipo son de nivel 02. Esto quiere decir que tienen un nivel por encima, el 01, o sea que pertenecen a domicilio. Esta forma lógica de organizar las variables es muy útil a la hora de diseñar un programa. Como dijimos, domicilio está formado por tipo, nombre y número, bien. Ahora vamos a fijarnos en una variable de segundo nivel como por ejemplo nombre. Esta variable es de tipo alfanumérico declarada con una longitud de 20 caracteres, ya sean letras o números. La X denota el tipo de dato: X indica que es alfanumérico y un 9 indica que es numérico (solo números). La variable número es de este último tipo y tiene una longitud de hasta 4 números, es decir, puede representar hasta el 9999.

Por otra parte si nos fijamos en TIPO vemos que está declarado como XX, esto denota que TIPO es del tipo alfanumérico y tiene una longitud de 2 caracteres. La variable NOMBRE también podría haberse definido como veinte ‘X’ consecutivas, pero es obvio el porqué no se ha definido así. COBOL puede hacer estas dos declaraciones que indican exactamente lo mismo, también es válido para los tipos numéricos.

Podríamos dedicar páginas a la explicación de los tipos de declaraciones de variables en Cobol, pero no es el ámbito de nuestro estudio, puede consultar los manuales en la bibliografía.

Una vez explicada la Working-Storage section solo nos quedarían por explicar la Linkage section, Comunication section y la Screen section.

La **Linkage section** sigue las mismas reglas de definición de variables que la Working-Storage section, solo que esta sección se usa para la comunicación con otros programas que se llamarían desde nuestro programa principal. Es muy habitual que en esta sección se incluyan definiciones de datos o también llamadas COPY con la directiva INCLUDE.

Las COPY son definiciones de datos que principalmente se usan para comunicarse con las rutinas que se invocan desde nuestro programa principal para informar los argumentos de entrada a la rutina y recoger los datos que la misma pudiera retornar.
También se usa para definir tablas DB2 y el área de comunicación entre el interprete de DB2 y el programa COBOL. El SQLCA (SQL Communication Area).

El SQLCA (figura 0-8) es una definición de estructura de datos que sirve para retornar información acerca del estado de la ejecución de consultas por parte del embebido de SQL que proporciona COBOL. No es más que una definición de variables con la misma sintaxis que la vista en la working-storage section. Es por así decirlo, la COPY que utiliza el interprete de DB2 para comunicarse con nuestro programa principal cuando invoca algún tipo de consulta de DB2 o de cualquier gestor de base de datos SQL. A continuación ilustramos el SQLCA en la figura 0-8.

```
01 SQLCA.
   05 SQLCAID        PIC X(8). VALUE X"0000000000000000".
   05 SQLCABC        PIC S9(9) BINARY.
   05 SQLCODE        PIC S9(9) BINARY.
   05 SQLERRM.
      49 SQLERRML    PIC S9(4) BINARY.
      49 SQLERRMC    PIC X(70).
   05 SQLERRP        PIC X(8).
   05 SQLERRD        OCCURS 6 TIMES
                     PIC S9(9) BINARY.
   05 SQLWARN.
      10 SQLWARN0    PIC X.
      10 SQLWARN1    PIC X.
      10 SQLWARN2    PIC X.
      10 SQLWARN3    PIC X.
      10 SQLWARN4    PIC X.
      10 SQLWARN5    PIC X.
      10 SQLWARN6    PIC X.
      10 SQLWARN7    PIC X.
      10 SQLWARN8    PIC X.
      10 SQLWARN9    PIC X.
      10 SQLWARNA    PIC X.
   05 SQLSTATE       PIC X(5).
```

Figura 0-8: Declaración en COBOL del SQLCA

Comunication section y Screen section
Comunication section Basta con decir que actualmente prácticamente no se usa y está obsoleta y la Screen Section es la encargada de almacenar información relativa a pantallas que se pueden generar en COBOL.

Con esto hemos terminado de exponer la Data division, ahora vamos a ver la que es posiblemente la división con más peso de todas en un programa COBOL, la Procedure división.

Procedure división (figura 0-9)
Aquí es donde se encuentran todas las instrucciones que ejecuta el programa y es aquí donde se implementa toda la lógica del programa.

No vamos a entrar todavía en detalles de comandos ni tipos de operaciones que se pueden hacer en COBOL, lo que vamos a exponer básicamente es la estructura del programa.
Un programa se suele organizar por párrafos, es decir, como en un documento escrito se van contando cosas, en un programa COBOL también se van "contando" cosas. Los párrafos se suelen agrupar por funcionalidad o por características que tengan en común.

Como en un documento cuando se redacta, se suelen agrupar las partes de la historia que tienen relación en un mismo capítulo, aquí también. Esta forma de programar es una de las buenas prácticas que se debería hacer (y se hace) a la hora de programar un lenguaje COBOL. Esto hace de COBOL un lenguaje bastante fácil de leer y seguir.

```
000123         *-----------------------------------
000124          PROCEDURE DIVISION.
000125         *-----------------------------------
000126         *MAIN SECTION.
000127                 PERFORM 1000-INICIO
000128                     THRU 1000-INICIO-EX.
000129         *
000130                 PERFORM 2000-PROCESO
000131                     THRU 2000-PROCESO-EX.
000132         *
000133                 PERFORM 3000-FIN
000134                     THRU 3000-FIN-EX.
000135         *
000136                 STOP RUN.
```

Figura 0-9: Ejemplo de Procedure Division

En la anterior imagen observamos que la división de proceso está estructurada en tres partes, inicio, proceso y final. Con la Instrucción PERFORM, entre otras cosas, se consigue llamar a los diferentes párrafos de nuestro programa, con lo cual nuestro programa primero llamará al párrafo 1000-INICIO, posteriormente al párrafo 2000-PROCESO y por ultimo al párrafo 3000-FINAL.

La directiva THRU que observamos es la función que se ejecuta justo cuando termina el párrafo al que hemos llamado, es el método que ejecutaremos después de llamar al párrafo, en general se suele implementar para devolver el control al punto donde se le llamó para que siga con la ejecución en un orden lógico.

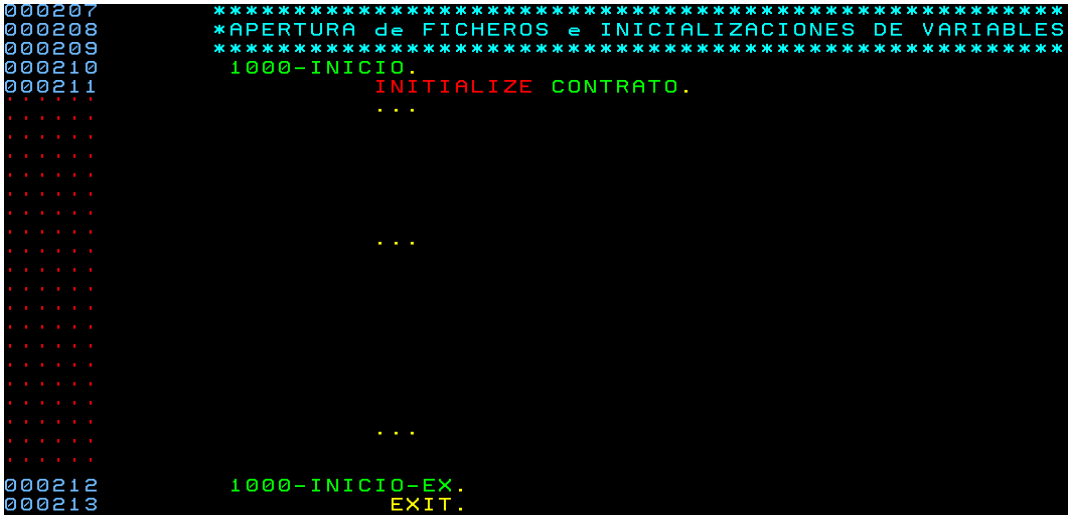


Figura 0-10: Ejemplo de esqueleto de un párrafo COBOL

El párrafo inicio (figura 0-10) suele contener inicializaciones de variables, apertura de ficheros y en definitiva, preparar el contexto del programa y sus datos para posteriormente ser utilizados en el párrafo 2000-PROCESO.

El párrafo 2000-PROCESO suele contener toda la lógica del programa, las operaciones de procesamiento y de cálculo. Por último, el párrafo 3000-FIN se suele utilizar para cerrar ficheros y asegurar la finalización correcta del programa. Si desea saber más sobre instrucciones y el lenguaje COBOL consulte las referencias [6] y [7].

J JCLs de compilación

Nombre del JCL: **COBOL2J**
Descripción: JCL de compilación del programa **COBOL2**

```
//KC025201 JOB ,
// MSGCLASS=H,MSGLEVEL=(1,1),TIME=(,4),REGION=144M,COND=(16,LT)
//*
//STP0000 EXEC PROC=ELAXFCOC
//* CICS=,
//* DB2=,
//* COMP=
//COBOL.SYSLIN DD DISP=SHR,
//         DSN=KC02520.LANG.OBJ(COBOL2)
//COBOL.SYSDEBUG DD DISP=SHR,
//         DSN=KC02520.COBOL.SYSDEBUG(COBOL2)
//COBOL.SYSLIB DD DISP=SHR,
//         DSN=KC02520.COBOL.SRC
//COBOL.SYSXMLSD DD DUMMY
//COBOL.SYSIN DD DISP=SHR,
//         DSN=KC02520.COBOL.SRC(COBOL2)
//*
//******* PASO DE ENLAZADO ******
//LKED EXEC PROC=ELAXFLNK
//LINK.SYSLIN DD *
     INCLUDE OBJ0000
/*
//LINK.SYSLMOD   DD  DISP=SHR,
//         DSN=KC02520.LANG.LOAD(COBOL2)
//LINK.OBJ0000 DD DISP=SHR,
//         DSN=KC02520.LANG.OBJ(COBOL2)
//*
//******* PASO DE LANZADO DEL PROGRAMA ******
//GO    EXEC   PROC=ELAXFGO,GO=COBOL2,
//            LOADDSN=KC02520.LANG.LOAD
//*SYSUT1   DD  DSN=KC02520.FILE1,DISP=(NEW,CATLG,DELETE),
//*             SPACE=(CYL,1000),DCB=(RECFM=FB,LRECL=20,BLKSIZE=0),
//*             UNIT=SYSDA
//SYSPRINT DD SYSOUT=*
//
```

Nombre: **PROGCJ**
Descripción: JCL de compilación del programa **PROGC**

```
//KC025201 JOB ,
// MSGCLASS=H,MSGLEVEL=(1,1),TIME=(,4),REGION=144M,COND=(16,LT)
//*
//STP0000 EXEC PROC=ELAXFCPC
//C.SYSLIN DD DISP=SHR,
//         DSN=KC02520.C.OBJ(PROGC)
//C.SYSLIB DD DISP=SHR,
//         DSN=CEE.SCEEH.H
//         DD DISP=SHR,
//         DSN=CEE.SCEEH.SYS.H
//         DD DISP=SHR,
//         DSN=CEE.SCEEH.NET.H
//         DD DISP=SHR,
//         DSN=CEE.SCEEH.NETINET.H
//C.SYSEVENT DD DUMMY
//C.SYSIN DD DISP=SHR,
//         DSN=KC02520.C.SRC(PROGC)
//*
//OPTION DD DATA,DLM='@@'
@@
//******* ADDITIONAL JCL FOR COMPILE HERE ******
//LKED EXEC PROC=ELAXFLNK
//LINK.OBJ0000 DD DISP=SHR,
//         DSN=KC02520.C.OBJ(PROGC)
//LINK.SYSLIN DD *
     INCLUDE OBJ0000
/*
//LINK.SYSLMOD   DD  DISP=SHR,
//         DSN=KC02520.C.LOAD(PROGC)
//*
//******* ADDITIONAL JCL FOR LINK HERE ******
//GO    EXEC   PROC=ELAXFGO,GO=PROGC,
//           LOADDSN=KC02520.C.LOAD
//******* ADDITIONAL RUNTIME JCL HERE ******
//
```

Nombre del JCL: **BATECOBJ**
Descripción: JCL de compilación del programa **BATECOB**

```
//KC025201 JOB ,
// MSGCLASS=H,MSGLEVEL=(1,1),TIME=(,30),REGION=144M,COND=(16,LT)
//*
//*
//*************************** PASO DE BORRADO ************************
//BORRADO   EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=*
//SYSIN DD *
       DEL KC02520L.FILEPRU
       SET MAXCC = 0
//*************************** PASO DE COMPILACION ********************
//STP0000 EXEC PROC=ELAXFCOC
//COBOL.SYSLIN DD DISP=SHR,
//        DSN=KC02520.LANG.OBJ(BATECOB)
//COBOL.SYSDEBUG DD DISP=SHR,
//        DSN=KC02520.COBOL.SYSDEBUG(BATECOB)
//COBOL.SYSLIB DD DISP=SHR,
//        DSN=KC02520.COBOL.SRC
//COBOL.SYSXMLSD DD DUMMY
//COBOL.SYSIN DD DISP=SHR,
//        DSN=KC02520.COBOL.SRC(BATECOB)
//*
//*************************** PASO DE ENLAZADO DE PROGRAMA *************
//LKED EXEC PROC=ELAXFLNK
//LINK.SYSLIN DD *
     INCLUDE OBJ0000
//LINK.SYSLMOD   DD  DISP=SHR,
//        DSN=KC02520.LANG.LOAD(BATECOB)
//LINK.OBJ0000 DD DISP=SHR,
//        DSN=KC02520.LANG.OBJ(BATECOB)
//*
//*************************** PASO DE EJECUCION **********************
//GO    EXEC   PROC=ELAXFGO,GO=BATECOB,
//           LOADDSN=KC02520.LANG.LOAD
//SYSUT1   DD  DSN=KC02520L.FILEPRU,DISP=(NEW,CATLG,DELETE),
//             SPACE=(CYL,1000),DCB=(RECFM=FB,LRECL=2,BLKSIZE=0),
//             UNIT=SYSDA
//SYSPRINT DD SYSOUT=*
//
```

Nombre del JCL: **BATECJ**
Descripción: JCL de compilación del programa **BATEC**

```
//KC025201 JOB ,
//  MSGCLASS=H,MSGLEVEL=(1,1),TIME=(,30),REGION=144M,COND=(16,LT)
//************************** PASO DE COMPILACION **********************
//STP0000 EXEC PROC=ELAXFCPC
//C.SYSCPRT DD DISP=SHR,
//         DSN=KC02520.PRUEBA.TXT(BATEC)
//C.SYSLIN DD DISP=SHR,
//         DSN=KC02520.C.OBJ(BATEC)
//C.SYSLIB DD DISP=SHR,
//         DSN=CEE.SCEEH.H
//         DD DISP=SHR,
//         DSN=CEE.SCEEH.SYS.H
//         DD DISP=SHR,
//         DSN=CEE.SCEEH.NET.H
//         DD DISP=SHR,
//         DSN=CEE.SCEEH.NETINET.H
//C.SYSEVENT DD DUMMY
//C.SYSIN DD DISP=SHR,
//         DSN=KC02520.C.SRC(BATEC)
//*
//OPTION DD DATA,DLM='@@'
@@
//************************** PASO DE ENLAZADO *************************
//LKED EXEC PROC=ELAXFLNK
//LINK.OBJ0000 DD DISP=SHR,
//         DSN=KC02520.C.OBJ(BATEC)
//LINK.SYSLIN DD *
     INCLUDE OBJ0000
//LINK.SYSLMOD   DD  DISP=SHR,
//         DSN=KC02520.C.LOAD(BATEC)
//*
//************************** PASO DE EJECUCION ************************
//GO    EXEC   PROC=ELAXFGO,GO=BATEC,
//            LOADDSN=KC02520.C.LOAD
//
```

- Nombre: **INSPROJJ**
- Descripción: JCL de compilación del programa **INSPROJ**

```
//KC025201 JOB ,
//  MSGCLASS=H,MSGLEVEL=(1,1),TIME=(,50),COND=(16,LT)
//*************************** PASO DE PRECOMPILACION ******************
//P0000 EXEC PROC=ELAXFCOP,REGION=0M,
//           DBRMDSN=KC02520.DB2DBRM,
//           DBRMMEM=INSPROJ
//COBPRE.SYSLIB DD DSN=DSNA10.SDSNLOAD,DISP=SHR
//COBPRE.SYSIN DD DISP=SHR,
//         DSN=KC02520.COBOL.SRC(INSPROJ)
//*
//*************************** PASO DE COMPILACION *********************
//STP0000 EXEC PROC=ELAXFCOC
//* CICS=,
//* DB2=,
//* COMP=
//COBOL.SYSLIN DD DISP=SHR,
//         DSN=KC02520.LANG.OBJ(INSPROJ)
//COBOL.SYSDEBUG DD DISP=SHR,
//         DSN=KC02520.COBOL.SYSDEBUG(INSPROJ)
//COBOL.SYSLIB DD DISP=SHR,
//         DSN=KC02520.COBOL.SRC
//COBOL.SYSXMLSD DD DUMMY
//COBOL.SYSIN    DD  DISP=(OLD,DELETE),DSN=&&DSNHOUT
//*************************** PASO DE ENLAZADO ************************
//LKED EXEC PROC=ELAXFLNK
//LINK.SYSLIN DD *
     INCLUDE OBJ0000
/*
//LINK.SYSLMOD   DD  DISP=SHR,
//         DSN=KC02520.LANG.LOAD(INSPROJ)
//LINK.OBJ0000 DD DISP=SHR,
//         DSN=KC02520.LANG.OBJ(INSPROJ)
//*
//*************************** PASO DE BIND ****************************
//BIND EXEC PGM=IKJEFT01
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//DBRMLIB  DD  DSN=KC02520.DB2DBRM(INSPROJ),DISP=SHR
//SYSTSIN   DD  *
     DSN SYSTEM(DBAG)
     BIND PACKAGE(DSN81010)-
     OWNER(KC02520) -
     MEMBER(INSPROJ) -
     ENCODING(EBCDIC) -
     LIBRARY('KC02520.DB2DBRM') -
     ACTION(REP) -
     VALIDATE(BIND)
     BIND PLAN(DBPRUD) -
     ACTION(REP) -
     ENCODING(EBCDIC) -
     PKLIST(DSN81010.*)
//*************************** PASO DE EJECUCION ***********************
//*********************************************************************
//* RUN PGM
//*********************************************************************
//RUNPGM   EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT),REGION=0M
//STEPLIB  DD DSN=DSNA10.SDSNLOAD,DISP=SHR
```

```
//         DD DSN=KC02520.LANG.LOAD,DISP=SHR
//DBRMLIB  DD DSN=KC02520.DB2DBRM(INSPROJ),DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSTSIN DD *
          DSN SYSTEM(DBAG)
          RUN PROGRAM(INSPROJ) PLAN(DBPRUD)
//
```

- Nombre: **INSEMPJ**
- Descripción: JCL de compilación del programa **INSEMP**

```
//KC025201 JOB ,
//  MSGCLASS=H,MSGLEVEL=(1,1),TIME=(,50),COND=(16,LT)
//*************************** PASO DE PRECOMPILACION ******************
//P0000 EXEC PROC=ELAXFCOP,REGION=0M,
//           DBRMDSN=KC02520.DB2DBRM,
//           DBRMMEM=INSEMP
//COBPRE.SYSLIB DD DSN=DSNA10.SDSNLOAD,DISP=SHR
//COBPRE.SYSIN DD DISP=SHR,
//         DSN=KC02520.COBOL.SRC(INSEMP)
//*
//*************************** PASO DE COMPILACION *********************
//STP0000 EXEC PROC=ELAXFCOC
//* CICS=,
//* DB2=,
//* COMP=
//COBOL.SYSLIN DD DISP=SHR,
//         DSN=KC02520.LANG.OBJ(INSEMP)
//COBOL.SYSDEBUG DD DISP=SHR,
//         DSN=KC02520.COBOL.SYSDEBUG(INSEMP)
//COBOL.SYSLIB DD DISP=SHR,
//         DSN=KC02520.COBOL.SRC
//COBOL.SYSXMLSD DD DUMMY
//COBOL.SYSIN    DD  DISP=(OLD,DELETE),DSN=&&DSNHOUT
//*************************** PASO DE ENLAZADO ************************
//LKED EXEC PROC=ELAXFLNK
//LINK.SYSLIN DD *
     INCLUDE OBJ0000
/*
//LINK.SYSLMOD   DD  DISP=SHR,
//         DSN=KC02520.LANG.LOAD(INSEMP)
//LINK.OBJ0000 DD DISP=SHR,
//         DSN=KC02520.LANG.OBJ(INSEMP)
//*
//*************************** PASO DE BIND ****************************
//BIND EXEC PGM=IKJEFT01
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//DBRMLIB  DD  DSN=KC02520.DB2DBRM(INSEMP),DISP=SHR
//SYSTSIN   DD  *
     DSN SYSTEM(DBAG)
     BIND PACKAGE(DSN81010)-
     OWNER(KC02520) -
     MEMBER(INSEMP) -
     ENCODING(EBCDIC) -
     LIBRARY('KC02520.DB2DBRM') -
```

```
     ACTION(REP) -
     VALIDATE(BIND)
     BIND PLAN(DBPRUD) -
     ACTION(REP) -
     ENCODING(EBCDIC) -
     PKLIST(DSN81010.*)
//*************************** PASO DE EJECUCION ***********************
//*********************************************************************
//* RUN PGM
//*********************************************************************
//RUNPGM   EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT),REGION=0M
//STEPLIB  DD DSN=DSNA10.SDSNLOAD,DISP=SHR
//         DD DSN=KC02520.LANG.LOAD,DISP=SHR
//DBRMLIB  DD DSN=KC02520.DB2DBRM(INSEMP),DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSTSIN DD *
          DSN SYSTEM(DBAG)
          RUN PROGRAM(INSEMP) PLAN(DBPRUD)
//
```

- Nombre: **INSEMACJ**
- Descripción: JCL de compilación del programa **INSEMAC**

```
//KC025201 JOB ,
// MSGCLASS=H,MSGLEVEL=(1,1),TIME=(,50),COND=(16,LT)
//*************************** PASO DE PRECOMPILACION ******************
//P0000 EXEC PROC=ELAXFCOP,REGION=0M,
//          DBRMDSN=KC02520.DB2DBRM,
//          DBRMMEM=INSEMAC
//COBPRE.SYSLIB DD DSN=DSNA10.SDSNLOAD,DISP=SHR
//COBPRE.SYSIN DD DISP=SHR,
//        DSN=KC02520.COBOL.SRC(INSEMAC)
//*
//*************************** PASO DE COMPILACION *********************
//STP0000 EXEC PROC=ELAXFCOC
//* CICS=,
//* DB2=,
//* COMP=
//COBOL.SYSLIN DD DISP=SHR,
//        DSN=KC02520.LANG.OBJ(INSEMAC)
//COBOL.SYSDEBUG DD DISP=SHR,
//        DSN=KC02520.COBOL.SYSDEBUG(INSEMAC)
//COBOL.SYSLIB DD DISP=SHR,
//        DSN=KC02520.COBOL.SRC
//COBOL.SYSXMLSD DD DUMMY
//COBOL.SYSIN    DD  DISP=(OLD,DELETE),DSN=&&DSNHOUT
//*************************** PASO DE ENLAZADO ************************
//LKED EXEC PROC=ELAXFLNK
//LINK.SYSLIN DD *
     INCLUDE OBJ0000
/*
//LINK.SYSLMOD   DD  DISP=SHR,
//        DSN=KC02520.LANG.LOAD(INSEMAC)
//LINK.OBJ0000 DD DISP=SHR,
//        DSN=KC02520.LANG.OBJ(INSEMAC)
//*
```

```
//*************************** PASO DE BIND ***************************
//BIND EXEC PGM=IKJEFT01
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//DBRMLIB  DD  DSN=KC02520.DB2DBRM(INSEMAC),DISP=SHR
//SYSTSIN   DD  *
     DSN SYSTEM(DBAG)
     BIND PACKAGE(DSN81010)-
     OWNER(KC02520) -
     MEMBER(INSEMAC) -
     ENCODING(EBCDIC) -
     LIBRARY('KC02520.DB2DBRM') -
     ACTION(REP) -
     VALIDATE(BIND)
     BIND PLAN(DBPRUD) -
     ACTION(REP) -
     ENCODING(EBCDIC) -
     PKLIST(DSN81010.*)
//*************************** PASO DE EJECUCION **********************
//********************************************************************
//* RUN PGM
//********************************************************************
//RUNPGM   EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT),REGION=0M
//STEPLIB  DD DSN=DSNA10.SDSNLOAD,DISP=SHR
//         DD DSN=KC02520.LANG.LOAD,DISP=SHR
//DBRMLIB  DD DSN=KC02520.DB2DBRM(INSEMAC),DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSTSIN DD *
         DSN SYSTEM(DBAG)
         RUN PROGRAM(INSEMAC) PLAN(DBPRUD)
//
```

K Expansión de los procesos ELAXFCOC, ELAXFLNK y ELAXFGO

```
//KC025201 JOB ,
JOB00663
         // MSGCLASS=H,MSGLEVEL=(1,1),TIME=(,4),REGION=144M,COND=(16,LT)
         //*
       2 //STP0000 EXEC PROC=ELAXFCOC
       3 XXELAXFCOC PROC LNGPRFX='IGY510',
         XX*           CICPRFX='DFH320.CICS',
         XX*           DB2PRFX='DSNA10',
         XX*           LODPRFX='FEK900'
         XX            LIBPRFX='CEE'
         XX*
       4 XXCOBOL  EXEC PGM=IGYCRCTL,REGION=0M
00340000
       5 XXSTEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,DISP=SHR
00350000
         IEFC653I SUBSTITUTION JCL - DSNAME=IGY510.SIGYCOMP,DISP=SHR
       6 XX         DD  DSNAME=&LIBPRFX..SCEERUN,DISP=SHR
00360000
         IEFC653I SUBSTITUTION JCL - DSNAME=CEE.SCEERUN,DISP=SHR
       7 XX         DD  DSNAME=&LIBPRFX..SCEERUN2,DISP=SHR
00370000
         IEFC653I SUBSTITUTION JCL - DSNAME=CEE.SCEERUN2,DISP=SHR
       8 XXSYSPRINT DD  SYSOUT=*
00380000
       9 //COBOL.SYSLIN DD DISP=SHR,
         //        DSN=KC02520.LANG.OBJ(COBOL2)
         X/SYSLIN   DD  DSNAME=&&LOADSET,UNIT=SYSALLDA,
00390000
         X/             DISP=(MOD,PASS),SPACE=(CYL,(1,1))
00400000
      10 XXSYSUT1   DD  UNIT=SYSALLDA,SPACE=(CYL,(1,1))
00420000
      11 XXSYSUT2   DD  UNIT=SYSALLDA,SPACE=(CYL,(1,1))
00430000
      12 XXSYSUT3   DD  UNIT=SYSALLDA,SPACE=(CYL,(1,1))
00440000
      13 XXSYSUT4   DD  UNIT=SYSALLDA,SPACE=(CYL,(1,1))
00450000
      14 XXSYSUT5   DD  UNIT=SYSALLDA,SPACE=(CYL,(1,1))
00460000
      15 XXSYSUT6   DD  UNIT=SYSALLDA,SPACE=(CYL,(1,1))
00470000
      16 XXSYSUT7   DD  UNIT=SYSALLDA,SPACE=(CYL,(1,1))
00480000
      17 XXSYSUT8   DD  UNIT=SYSALLDA,SPACE=(CYL,(1,1))
00490000
      18 XXSYSUT9   DD  UNIT=SYSALLDA,SPACE=(CYL,(1,1))
00500000
      19 XXSYSUT10  DD  UNIT=SYSALLDA,SPACE=(CYL,(1,1))
00510000
      20 XXSYSUT11  DD  UNIT=SYSALLDA,SPACE=(CYL,(1,1))
00520000
      21 XXSYSUT12  DD  UNIT=SYSALLDA,SPACE=(CYL,(1,1))
00530000
```

```
       22 XXSYSUT13  DD  UNIT=SYSALLDA,SPACE=(CYL,(1,1))
00540000
       23 XXSYSUT14  DD  UNIT=SYSALLDA,SPACE=(CYL,(1,1))
00550000
       24 XXSYSUT15  DD  UNIT=SYSALLDA,SPACE=(CYL,(1,1))
00560000
       25 XXSYSMDECK DD  UNIT=SYSALLDA,SPACE=(CYL,(1,1))
00565000
          //* CICS=,
          //* DB2=,
          //* COMP=
       26 //COBOL.SYSDEBUG DD DISP=SHR,
          //        DSN=KC02520.COBOL.SYSDEBUG(COBOL2)
       27 //COBOL.SYSLIB DD DISP=SHR,
          //        DSN=KC02520.COBOL.SRC
       28 //COBOL.SYSXMLSD DD DUMMY
       29 //COBOL.SYSIN DD DISP=SHR,
          //        DSN=KC02520.COBOL.SRC(COBOL2)
          //*
          //******* ADDITIONAL JCL FOR COMPILE HERE ******
       30 //LKED EXEC PROC=ELAXFLNK
       31 XXELAXFLNK PROC BIND='HEWL',
          XX            CICPRFX='DFH320.CICS',
          XX            DB2PRFX='DSNA10',
          XX            LIBPRFX='CEE'
          XX*
       32 XXLINK     EXEC PGM=&BIND,COND=(8,LT),REGION=1024K,
          XX            PARM=('MAP')
          IEFC653I SUBSTITUTION JCL -
PGM=HEWL,COND=(8,LT),REGION=1024K,PARM=('MAP')
       33 XXSYSLIB   DD DISP=SHR,
          XX            DSN=&LIBPRFX..SCEELKED
          IEFC653I SUBSTITUTION JCL - DISP=SHR,DSN=CEE.SCEELKED
       34 XX         DD DISP=SHR,
          XX            DSN=&CICPRFX..SDFHLOAD
          IEFC653I SUBSTITUTION JCL - DISP=SHR,DSN=DFH320.CICS.SDFHLOAD
       35 XX         DD DISP=SHR,
          XX            DSN=&DB2PRFX..SDSNLOAD
          IEFC653I SUBSTITUTION JCL - DISP=SHR,DSN=DSNA10.SDSNLOAD
       36 //LINK.SYSLIN DD *
          X/SYSLIN   DD DUMMY
       37 //LINK.SYSLMOD   DD  DISP=SHR,
          //        DSN=KC02520.LANG.LOAD(COBOL2)
          X/SYSLMOD  DD DUMMY
       38 XXSYSPRINT DD SYSOUT=*
       39 XXSYSUT1   DD UNIT=SYSALLDA,SPACE=(TRK,(10,10))
          XX*
       40 //LINK.OBJ0000 DD DISP=SHR,
          //        DSN=KC02520.LANG.OBJ(COBOL2)
          //*
          //******* ADDITIONAL JCL FOR LINK HERE ******
          //******* ADDITIONAL JCL FOR LINK HERE ******
       41 //GO    EXEC   PROC=ELAXFGO,GO=COBOL2,
          //           LOADDSN=KC02520.LANG.LOAD
       42 XXELAXFGO  PROC LIBPRFX='CEE',
          XX            LODPRFX='FEK900'
          XX*
       43 XXRUN      EXEC PGM=&GO,COND=(4,LT),REGION=0M
          IEFC653I SUBSTITUTION JCL - PGM=COBOL2,COND=(4,LT),REGION=0M
       44 XXSTEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,DISP=SHR
          IEFC653I SUBSTITUTION JCL - DSNAME=CEE.SCEERUN,DISP=SHR
```

```
        45 XX         DD  DSNAME=&LIBPRFX..SCEERUN2,DISP=SHR
           IEFC653I SUBSTITUTION JCL - DSNAME=CEE.SCEERUN2,DISP=SHR
        46 XX         DD DISP=SHR,
           XX            DSN=&LODPRFX..SFEKLOAD
           IEFC653I SUBSTITUTION JCL - DISP=SHR,DSN=FEK900.SFEKLOAD
        47 XX         DD DISP=SHR,
           XX            DSN=&LOADDSN
           IEFC653I SUBSTITUTION JCL - DISP=SHR,DSN=KC02520.LANG.LOAD
        48 //SYSPRINT DD SYSOUT=*
           X/SYSPRINT DD SYSOUT=*
        49 XXSORTWK01 DD  UNIT=SYSDA,SPACE=(1024,(100,100))
        50 XXSORTWK02 DD  UNIT=SYSDA,SPACE=(1024,(100,100))
        51 XXSORTWK03 DD  UNIT=SYSDA,SPACE=(1024,(100,100))
        52 XXSYSOUT   DD  SYSOUT=*
        53 XXSYSPRINT DD  SYSOUT=*
        54 XXCEEDUMP  DD  SYSOUT=*
        55 XXSYSUDUMP DD  DUMMY
        56 XXSYSPRINT DD SYSOUT=*
           XX*
           //*SYSUT1   DD  DSN=KC02520.FILE1,DISP=(NEW,CATLG,DELETE),
           //*
SPACE=(CYL,1000),DCB=(RECFM=FB,LRECL=20,BLKSIZE=0),
           //*             UNIT=SYSDA
```

Como podemos observar en el código anterior, en negrita se han resaltado los procesos ELAXFCOC, ELAXFLNK y ELAXFGO y sus consiguientes sustituciones en el JCL.

- Para el proceso **ELAXFCOC** observamos que se ejecuta el **programa IGYCRCTL**, este es el verdadero compilador de COBOL en el mainframe.

- Para el proceso **ELAXFLNK** se utiliza el **programa HEWL**, este programa se encarga de formar el ejecutable.

- Para el proceso **ELAXFGO** observamos que lo único que se hace es llamar al ejecutable que creó **HEWL**.

L Programas

Nombre del Programa: **COBOL2**
Descripción: Programa que abre un fichero, si lo consigue lo cierra.

```
000100*****************************************************************
00010004
000200*  Programa que abre un fichero y si lo consigue, lo cierra     *
00020004
000300*                                                               *
00030004
000500*****************************************************************
00050004
000600*****************************************************************
00060004
000700**           I D          D I V I S I O N                    ***
00070004
000800*****************************************************************
00080004
000900 IDENTIFICATION DIVISION.
00090004
001000 PROGRAM-ID.    COBOL2.
00100004
001100*****************************************************************
00110004
001200**           D A T A      D I V I S I O N                    ***
00120004
001300*****************************************************************
00130004
001400 ENVIRONMENT DIVISION.
00140004
001500 CONFIGURATION SECTION.
00150004
001600 SOURCE-COMPUTER.        MAINFR.
00160004
001700 OBJECT-COMPUTER.        MAINFR.
00170004
001800 SPECIAL-NAMES.
00180004
001900
00190004
002000     DECIMAL-POINT IS COMMA.
00200004
002100
00210004
002200 INPUT-OUTPUT SECTION.
00220004
002300 FILE-CONTROL.
00230004
002400
00240004
002500        SELECT FICHERO1 ASSIGN to "FICHERO1"
00250004
002600        ACCESS MODE IS SEQUENTIAL
00260004
002700        FILE STATUS IS STATUS-FICHERO.
00270004
```

```
002800
00280004
003200
00320004
003300 DATA DIVISION.
00330004
003400
00340004
003500 FILE SECTION.
00350004
003600
00360004
003700  FD FICHERO1
00370004
003800      BLOCK CONTAINS 1 RECORDS
00380004
003900      LABEL RECORD ARE STANDARD
00390004
004000      RECORDING MODE IS F
00400004
004100      RECORD CONTAINS 2 CHARACTERS
00410004
004200      DATA RECORD IS REG.
00420004
004300  01  REG                        PIC X(2).
00430004
004310
00431004
004391
00439104
004392*
00439204
004393 WORKING-STORAGE SECTION.
00439304
004394 01 STATUS-FICHERO            PIC XX   VALUE "00".
00439404
004456
00445604
004457*-----------------------------------------------------------------
00445704
004458 PROCEDURE DIVISION.
00445804
004459*-----------------------------------------------------------------
00445904
004460*MAIN SECTION.
00446004
004461     PERFORM 100-OPEN-FILE.
00446104
004462     Stop Run.
00446204
004470**
00447004
004500** Abre El fichero y muestra el codigo de retorno
00450004
004600**
00460004
004700 100-OPEN-FILE.
00470004
004800     OPEN I-O FICHERO1
00480004
```

```
004810     IF STATUS-FICHERO = "00" THEN
00481004
004820        DISPLAY "FICHERO ABIERTO CORRECTAMENTE CON CODIGO "
00482004
004821        STATUS-FICHERO
00482104
004822        CLOSE FICHERO1
00482204
004830     ELSE
00483004
004840        DISPLAY "IMPOSIBLE ABRIR, ERROR: " STATUS-FICHERO
00484004
004850     END-IF.
00485004
004860 100-OPEN-FILE-EX.
00486004
004870           EXIT.
00487004
006900 End program COBOL2.
```

Nombre del Programa: **PROGC**
Descripción: Programa que abre un fichero y si lo consigue, lo cierra.

```
#include <stdio.h>
#include <stdlib.h>

int main(){
      FILE *fp;
      fp=NULL;
      fp=fopen("FICHERO1","r");

            if(fp==NULL){
                  printf("\nIMPOSIBLE ABRIR EL FICHERO1\n\n");
                  printf("retorno %c es un NULL\n", fp);
                  return -1;
            }
            printf("ABIERTO CORRECTAMENTE\n");
            printf("FIN DEL PROGRAMA\n");
            fclose(fp);
      return 0;

}
```

Nombre del Programa: **BATECOB**
Descripción: Programa que cuantifica el tiempo que tarda cada instrucción.

```
      000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID. BATECOB
000003*
000004*
000005* BATERIA DE PRUEBAS TEST DE RENDIMIENTO LENGUAJE COBOL
000006*
000007*
000008*
000009*-----------------------------------------------------------------
000010 ENVIRONMENT DIVISION.
000011 CONFIGURATION SECTION.
000012 SOURCE-COMPUTER.        MAINFR.
```

```
000013 OBJECT-COMPUTER.        MAINFR.
000014 SPECIAL-NAMES.
000015
000016         DECIMAL-POINT IS COMMA.
000017
000018 INPUT-OUTPUT SECTION.
000019 FILE-CONTROL.
000020
000021     SELECT FILEPRU ASSIGN TO SYSUT1
000022        ACCESS MODE IS SEQUENTIAL
000023        FILE STATUS IS WK-STATUS-FILEPRU.
000024
000025
000026 DATA DIVISION.
000027
000028 FILE SECTION.
000029
000030 FD FILEPRU
000031     BLOCK CONTAINS 0 RECORDS
000032     LABEL RECORD ARE STANDARD
000033     RECORDING MODE IS F
000034     RECORD CONTAINS 2 CHARACTERS
000035     DATA RECORD IS REG.
000036 01  REG                            PIC X(2).
000037
000038
000039 WORKING-STORAGE SECTION.
000040***************** VARIABLES AUXILIARES *************************
000041 01  WK-VARIABLES.
000042     02  WK-STATUS-FILEPRU          PIC X(2)       VALUE '00'.
000043     02  WK-RESTO                   PIC 9(10)v9(7) VALUE ZEROS.
000044     02  WK-NUM-OPER                PIC 9(10)      VALUE ZEROS.
000045     02  WK-BIG                     PIC S9(9)V9(7) VALUE ZEROS.
000046
000047     02  WK-NUM-FICH                PIC 9(10)      VALUE ZEROS.
000048     02  WK-NUM-PRUE                PIC 9(2)       VALUE ZEROS.
000049
000050***************** COPY DE LOG DE IMPRESION *********************
000051 01  WK-LOG.
000052     02  WK-MSGCAB1                 PIC X(80).
000053     02  WK-MSGCAB2                 PIC X(80).
000054     02  WK-MSGCAB3                 PIC X(80).
000055     02  WK-MSG1                    PIC X(23).
000056     02  WK-HORA-TIME               PIC X(2).
000057     02  WK-MSG2                    PIC X(10).
000058     02  WK-MIN-TIME                PIC X(2).
000059     02  WK-MSG3                    PIC X(11).
000060     02  WK-SEC-TIME                PIC X(2).
000061     02  WK-MSG4                    PIC X(12).
000062     02  WK-CEN-TIME                PIC X(2).
000063     02  WK-MSG5                    PIC X(6).
000064     02  WK-INSTR                   PIC X(30).
000065     02  WK-MSG6                    PIC X(80).
000066
000067***************** VARIABLES TEMPORALES *************************
000068 01  WK-VAR-TIEMPO.
000069     02  WK-MARCA-INI               PIC 9(9)V9(2)  VALUE ZEROS.
000070     02  WK-MARCA-FIN               PIC 9(9)V9(2)  VALUE ZEROS.
000071     02  WK-MARCA                   PIC 9(9)V9(2)  VALUE ZEROS.
000072     02  WK-MARCA-INIF              PIC 9(9)V9(2)  VALUE ZEROS.
000073     02  WK-MARCA-FINF              PIC 9(9)V9(2)  VALUE ZEROS.
```

```
000074      02  WK-CALC-SEGUNDOS           PIC 9(9)       VALUE ZEROS.
000075      02  WK-CALC-CENTESIMAS         PIC 9(9)V9(2)  VALUE ZEROS.
000076      02  WK-CALC-MINUTOS            PIC 9(9)       VALUE ZEROS.
000077      02  WK-CALC-HORA               PIC 9(9)       VALUE ZEROS.
000078      02  WK-INSTRUCTION             PIC X(30)      VALUE SPACE.
000079      02  WK-MARCA-TMP.
000080          03  WK-MARCA-HORA          PIC 9(2)       VALUE ZEROS.
000081          03  WK-MARCA-MIN           PIC 9(2)       VALUE ZEROS.
000082          03  WK-MARCA-SEC           PIC 9(2)       VALUE ZEROS.
000083          03  WK-MARCA-CEN           PIC 9(2)       VALUE ZEROS.
000084****************** CONSTANTES NUMERICAS ************************
000085 01  CN-CONSTANTES.
000086      02  CN-BIG                     PIC S9(9)V9(7)
000087                                           VALUE 987654321,8554469.
000088      02  CN-NUM-OPER                PIC 9(10)      VALUE 1000000.
000089      02  CN-NUM-FICH                PIC 9(10)      VALUE 60000000.
000090      02  CN-NUM-TEST                PIC 9(02)      VALUE 8.
000091****************** CONSTANTES ALFANUMERICAS *********************
000092 01  CA-CONSTANTES.
000093      02  CONSTANTE                  PIC X(2)       VALUE 'AA'.
000094      02  STRING1                    PIC X(5)       VALUE 'PABLO'.
000095      02  STRING2                    PIC X(5)       VALUE 'PEREZ'.
000096      02  STRING-AUX                 PIC X(5)       VALUE SPACES.
000097
000098*-----------------------------------------------------------------
000099 PROCEDURE DIVISION.
000100*-----------------------------------------------------------------
000101*MAIN SECTION.
000102      PERFORM 1000-INICIO
000103         THRU 1000-INICIO-EXIT
000104*
000105      PERFORM 2000-PROCESO
000106         THRU 2000-PROCESO-EXIT
000107*
000108      PERFORM 3000-FIN
000109         THRU 3000-FIN-EXIT
000110*
000111      .
000112****************** DECLARACION DE PARRAFOS DEL PROGRAMA **********
000113******************************************************************
000114 1000-INICIO.
000115******************************************************************
000116* INICIALIZACION DE VARIABLES
000117******************************************************************
000118      INITIALIZE WK-LOG
000119                 WK-VARIABLES
000120                 WK-VAR-TIEMPO
000121*
000122      MOVE CN-NUM-TEST                    TO WK-NUM-PRUE
000123      MOVE CN-NUM-OPER                    TO WK-NUM-OPER
000124      MOVE CN-NUM-FICH                    TO WK-NUM-FICH
000125      MOVE '************************************************************
000126-     '************************' TO WK-MSGCAB1
000127      MOVE 'TIEMPO TRANSCURRIDO -->'          TO WK-MSG1
000128      MOVE ' MINUTOS: '                   TO WK-MSG2
000129      MOVE ' SEGUNDOS: '                  TO WK-MSG3
000130      MOVE ' CENTESIMAS '                 TO WK-MSG4
000131      MOVE ' PARA '                       TO WK-MSG5
000132
000133*
000134      .
```

```
000135 1000-INICIO-EXIT.
000136     EXIT
000137*
000138     .
000139*****************************************************************
000140 2000-PROCESO.
000141*****************************************************************
000142* PROCESO DE PROGRAMA
000143*****************************************************************
000144     PERFORM VARYING WK-NUM-PRUE FROM 1 BY 1
000145     UNTIL WK-NUM-PRUE > CN-NUM-TEST
000146
000147        PERFORM GET-MARCA-TIEMP
000148           THRU GET-MARCA-TIEMP-EXIT
000149
000150        MOVE WK-MARCA                       TO WK-MARCA-INI
000151
000152        IF WK-MARCA-INIF = ZEROS
000153           MOVE WK-MARCA                    TO WK-MARCA-INIF
000154        END-IF
000155
000156        IF WK-NUM-PRUE < 7
000157************ TEST DE OPERACIONES DE CALCULO **********************
000158           PERFORM OPERACIONES-CALCULO
000159              THRU OPERACIONES-CALCULO-EXIT
000160
000161        ELSE
000162************ TEST DE OPERACIONES CON FICHEROS ********************
000163           PERFORM OPERACIONES-FICHERO
000164              THRU OPERACIONES-FICHERO-EXIT
000165*
000166        END-IF
000167************ CALCULO TIEMPO FIN DE OPERACION *********************
000168
000169        PERFORM GET-MARCA-TIEMP
000170           THRU GET-MARCA-TIEMP-EXIT
000171
000172        MOVE    WK-MARCA                    TO WK-MARCA-FINF
000173
000174        PERFORM CALCULA-DURACION
000175           THRU CALCULA-DURACION-EXIT
000176*
000177     END-PERFORM
000178*
000179     .
000180 2000-PROCESO-EXIT.
000181     EXIT
000182*
000183     .
000184*
000185 OPERACIONES-CALCULO.
000186*
000187              EVALUATE WK-NUM-PRUE
000188                  WHEN 1
000189
000190                     MOVE 'COMPUTE-SUM'    TO WK-INSTRUCTION
000191                     MOVE CN-BIG           TO WK-BIG
000192                     PERFORM COMPUTE-SUM
000193                        THRU COMPUTE-SUM-EXIT
000194                     WK-NUM-OPER TIMES
000195
```

```
000196                   WHEN 2
000197
000198                      MOVE 'COMPUTE-RES'   TO WK-INSTRUCTION
000199                      MOVE CN-BIG          TO WK-BIG
000200                      PERFORM COMPUTE-RES
000201                         THRU COMPUTE-RES-EXIT
000202                      WK-NUM-OPER TIMES
000203
000204                   WHEN 3
000205
000206                      MOVE 'COMPUTE-DIV'   TO WK-INSTRUCTION
000207                      MOVE CN-BIG          TO WK-BIG
000208                      PERFORM COMPUTE-DIV
000209                         THRU COMPUTE-DIV-EXIT
000210                      WK-NUM-OPER TIMES
000211
000212                   WHEN 4
000213
000214                      MOVE 'COMPUTE-MUL'   TO WK-INSTRUCTION
000215                      MOVE CN-BIG          TO WK-BIG
000216                      PERFORM COMPUTE-MUL
000217                         THRU COMPUTE-MUL-EXIT
000218                      WK-NUM-OPER TIMES
000219
000220                   WHEN 5
000221
000222                      MOVE 'COMPUTE-POT'   TO WK-INSTRUCTION
000223                      MOVE CN-BIG          TO WK-BIG
000224                      PERFORM COMPUTE-POT
000225                         THRU COMPUTE-POT-EXIT
000226                      WK-NUM-OPER TIMES
000227
000228                   WHEN 6
000229
000230                      MOVE 'SWAP'          TO WK-INSTRUCTION
000231                      PERFORM OPER-SWAP
000232                         THRU OPER-SWAP-EXIT
000233                      WK-NUM-OPER TIMES
000234
000235               END-EVALUATE
000236*
000237     .
000238*
000239 OPERACIONES-CALCULO-EXIT.
000240     EXIT.
000241*
000242 COMPUTE-DIV.
000243*
000244     COMPUTE WK-BIG = CN-BIG / 123456,78.
000245*
000246 COMPUTE-DIV-EXIT.
000247     EXIT.
000248*
000249 COMPUTE-SUM.
000250*
000251     ADD 123456,78                        TO WK-BIG.
000252*
000253 COMPUTE-SUM-EXIT.
000254     EXIT.
000255*
000256 COMPUTE-RES.
```

```
000257*
000258     SUBTRACT 123456,78                    FROM WK-BIG.
000259*
000260 COMPUTE-RES-EXIT.
000261     EXIT.
000262*
000263 COMPUTE-MUL.
000264*
000265     COMPUTE WK-BIG = CN-BIG * 123456,78.
000266*
000267 COMPUTE-MUL-EXIT.
000268     EXIT.
000269*
000270 COMPUTE-POT.
000271*
000272     COMPUTE WK-BIG = CN-BIG ** 6,78.
000273*
000274 COMPUTE-POT-EXIT.
000275     EXIT.
000276*
000277 OPERACIONES-FICHERO.
000278*
000279     EVALUATE WK-NUM-PRUE
000280           WHEN 7
000281
000282              MOVE 'ESCRIBIR '           TO WK-INSTRUCTION
000283              PERFORM OPER-ESCRIBIR
000284                 THRU OPER-ESCRIBIR-EXIT
000285
000286           WHEN 8
000287
000288              MOVE 'LEER  '              TO WK-INSTRUCTION
000289              PERFORM OPER-LEER
000290                 THRU OPER-LEER-EXIT
000291
000292     END-EVALUATE
000293*
000294     .
000295*
000296 OPERACIONES-FICHERO-EXIT.
000297     EXIT.
000298*
000299 OPER-ESCRIBIR.
000300*
000301     OPEN OUTPUT FILEPRU.
000302*
000303     EVALUATE WK-STATUS-FILEPRU
000304        WHEN 00
000305*
000306           PERFORM WK-NUM-FICH         TIMES
000307              WRITE REG FROM CONSTANTE
000308              END-WRITE
000309           END-PERFORM
000310*
000311           CLOSE FILEPRU
000312        WHEN OTHER
000313*
000314           DISPLAY 'ERROR DE FICHERO ' WK-STATUS-FILEPRU
000315           STOP RUN
000316*
000317     END-EVALUATE
```

```
000318*
000319     .
000320*
000321 OPER-ESCRIBIR-EXIT.
000322     EXIT.
000323*
000324 OPER-LEER.
000325*
000326     OPEN INPUT FILEPRU
000327*
000328     EVALUATE WK-STATUS-FILEPRU
000329        WHEN 00
000330*
000331           PERFORM CN-NUM-FICH         TIMES
000332              READ FILEPRU INTO CONSTANTE
000333              END-READ
000334           END-PERFORM
000335*
000336           CLOSE FILEPRU
000337*
000338     WHEN OTHER
000339*
000340        DISPLAY 'ERROR DE FICHERO ' WK-STATUS-FILEPRU
000341        STOP RUN
000342*
000343     END-EVALUATE
000344*
000345     .
000346*
000347 OPER-LEER-EXIT.
000348     EXIT.
000349*
000350 OPER-SWAP.
000351*
000352     MOVE STRING1                          TO STRING-AUX
000353     MOVE STRING2                          TO STRING1
000354     MOVE STRING-AUX                       TO STRING2
000355*
000356     .
000357 OPER-SWAP-EXIT.
000358     EXIT.
000359*
000360 GET-MARCA-TIEMP.
000361*
000362     ACCEPT WK-MARCA-TMP FROM TIME.
000363*
000364     COMPUTE WK-MARCA ROUNDED = ( WK-MARCA-HORA * 3600 )  +
000365     ( WK-MARCA-MIN * 60    )  +
000366     WK-MARCA-SEC              +
000367     ( WK-MARCA-CEN / 100   )
000368*
000369     .
000370 GET-MARCA-TIEMP-EXIT.
000371     EXIT.
000372*****************************************************************
000373 3000-FIN.
000374*****************************************************************
000375* FINAL DE PROGRAMA
000376*****************************************************************
000377****************** CALCULO DE DURACION DEL PROGRAMA *************
000378     MOVE WK-MARCA-INIF                    TO WK-MARCA-INI
```

```
000379      MOVE WK-MARCA-FINF                    TO WK-MARCA
000380      MOVE 'TEST COMPLETADO'                TO WK-INSTRUCTION
000381*****
000382      PERFORM CALCULA-DURACION
000383         THRU CALCULA-DURACION-EXIT
000384*
000385      DISPLAY '********************************** FIN ************
000386-     '****************************'
000387      STOP RUN
000388*
000389      .
000390 3000-FIN-EXIT.
000391      EXIT
000392*
000393      .
000394*
000395 SET-MARCA-TIEMP.
000396*
000397      MOVE WK-MARCA                         TO WK-CALC-SEGUNDOS
000398      MOVE ZEROS                            TO WK-CALC-MINUTOS
000399*                                              WK-CALC-HORA
000400      DIVIDE WK-CALC-SEGUNDOS
000401      BY 60
000402      GIVING WK-CALC-MINUTOS
000403      REMAINDER WK-CALC-SEGUNDOS
000404*
000405      IF WK-CALC-MINUTOS > 60
000406         DIVIDE WK-CALC-MINUTOS BY 60
000407         GIVING WK-CALC-MINUTOS REMAINDER WK-CALC-HORA
000408      END-IF
000409*
000410      COMPUTE WK-CALC-CENTESIMAS = WK-CALC-SEGUNDOS - WK-MARCA
000411      COMPUTE WK-CALC-CENTESIMAS = WK-CALC-CENTESIMAS * 100
000412      COMPUTE WK-RESTO = WK-CALC-SEGUNDOS / 60
000413*
000414      MOVE WK-CALC-CENTESIMAS               TO WK-MARCA-CEN
000415      MOVE WK-CALC-HORA                     TO WK-MARCA-HORA
000416      MOVE WK-CALC-MINUTOS                  TO WK-MARCA-MIN
000417      MOVE WK-CALC-SEGUNDOS                 TO WK-MARCA-SEC
000418*
000419      .
000420 SET-MARCA-TIEMP-EXIT.
000421      EXIT.
000422*
000423 CALCULA-DURACION.
000424*
000425      MOVE WK-MARCA                         TO WK-MARCA-FIN
000426*
000427      COMPUTE WK-MARCA = WK-MARCA-FIN - WK-MARCA-INI
000428*
000429      PERFORM SET-MARCA-TIEMP
000430         THRU SET-MARCA-TIEMP-EXIT
000431*
000432      MOVE WK-MARCA-MIN                     TO WK-MIN-TIME
000433      MOVE WK-MARCA-SEC                     TO WK-SEC-TIME
000434      MOVE WK-MARCA-CEN                     TO WK-CEN-TIME
000435      MOVE WK-INSTRUCTION                   TO WK-INSTR
000436      DISPLAY WK-LOG
000437*
000438      .
000439*
```

```
000440 CALCULA-DURACION-EXIT.
000441     EXIT.
000442*
000443 END PROGRAM BATECOB.
```

Nombre del Programa: **BATEC**
Descripción: Programa que cuantifica el tiempo que tarda cada instrucción del lenguaje C.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

void imprimeTiempo(time_t comienzo, time_t final,int mod, char * s,
clock_t comienzoC){
       struct tm *tiempoComienzoPtr, *tiempoFinalPtr;
      tiempoComienzoPtr = gmtime( &comienzo );
      tiempoFinalPtr = gmtime( &final );

      if (mod==1){
            printf( "Número de segundos transcurridos desde el comienzo"
                        " del programa: %f s\n",(double)difftime(final,
comienzo) );
      }
      else
            printf( "Número de segundos transcurridos en "
                        "la operacion %s: %f s\n",s,
                       (clock()-comienzoC)/(double)CLOCKS_PER_SEC);
      return;
}
void leerFile(FILE *fp){
      char ab[2];
      fread( ab, sizeof(char), 2, fp );
      return;
}

void escribirFile(FILE *fp){
      char ab[2] = { 'a', 'b' };
      fwrite( ab, sizeof(char), 2, fp );
      return;
}

void ComputaDiv(){
      double a=987654321.8554469;
      a = a / 123456.78;
      return;
}

void ComputaSum(){
      double a=987654321.8554469;
      a = a + 123456.78;
      return;
}

void ComputaRes(){
      double a=987654321.8554469;
      a = a - 123456.78;
      return;
}
```

```
void ComputaMul(){
      double a=987654321.8554469;
      a = a * 123456.78;
      return;
}

void ComputaPot(){
      double a=987654321.8554469, b=6.78;
      a = pow(a,b);
      return;
}

void swap(int *a,int *b){
      int tmp=0;
      tmp=*a;
      *a=*b;
      *b=tmp;
      return;
}

int main(){
      time_t comienzo, final, comienzoT, finalT;
      struct tm *tiempoComienzoPtr, *tiempoFinalPtr;
      clock_t comienzoC;
      FILE *fp;
      int i=0,j=0,k=0,a=1,b=2;
      int numTest=9;
      int numComp=1000000;
      int    numF=60000000;
/********************** APERTURA DE FICHERO ************************/
      fp=NULL;
      fp=fopen("PRUEBA.TXT","w");
      if(fp==NULL){
            printf("ERROR de fichero\n");
            return -1;
      }
/************************ INICIO DE LOS TEST ************************/
      comienzo = time( NULL );
      for(i=0;i<numTest;i++){
            if(i<3){
                  comienzoT = time( NULL );
                  comienzoC=clock();
/************************ OPERACIONES DE FICHEROS *********************/
                  for(k=0;k<numF;k++)
                  {
                        if(i==1)
                              escribirFile(fp);
                        if(i==2)
                              leerFile(fp);
                  }
/********** IMPRESION DE LOS TIEMPOS DE OPERACION DE FICHEROS *********/
                  if(i==1){
                        finalT = time( NULL );
                        imprimeTiempo(comienzoT, finalT,2,"
ESCRITURA",comienzoC);
                  }
                  if(i==2){
                        finalT = time( NULL );
                        imprimeTiempo(comienzoT, finalT,2,"
LECTURA",comienzoC);
                  }
```

```
            }
/********************** OPERACIONES DE COMPUTO ********************/
            if(i>2){
                  comienzoT = time( NULL );
   comienzoC=clock();
                  for(j=0;j<numComp;j++){
                        if(i==3)
                              ComputaDiv();
                        if(i==4)
                              ComputaSum();
                        if(i==5)
                              ComputaRes();
                        if(i==6)
                              ComputaMul();
                        if(i==7)
                              ComputaPot();
                        if(i==8)
                              swap(&a,&b);
                  }

/********** IMPRESION DE LOS TIEMPOS DE OPERACION DE COMPUTO *********/
                  if(i==3){
                        finalT = time( NULL );
                        imprimeTiempo(comienzoT, finalT,2,"
DIVISION",comienzoC);
                  }
                  if(i==4){
                        finalT = time( NULL );
                        imprimeTiempo(comienzoT, finalT,2,"
SUMA",comienzoC);
                  }
                  if(i==5){
                        finalT = time( NULL );
                        imprimeTiempo(comienzoT, finalT,2,"
RESTA",comienzoC);
                  }
                  if(i==6){
                        finalT = time( NULL );
                        imprimeTiempo(comienzoT, finalT,2,"
MULTIPLICACION",comienzoC);
                  }
                  if(i==7){
                        finalT = time( NULL );
                        imprimeTiempo(comienzoT, finalT,2,"
POTENCIA",comienzoC);
                  }
                  if(i==8){
                        finalT = time( NULL );
                        imprimeTiempo(comienzoT, finalT,2,"
SWAP",comienzoC);
                  }
            }
/**********************FIN de los test*************************/

      }
      final = time( NULL );
      imprimeTiempo(comienzo, final,1,NULL,comienzoC);
      fclose (fp);

      return 0;
}
```

Nombre del Programa: **SELNOT**
Descripción: Programa que cuantifica el tiempo que tarda la consulta con doble NOT EXISTS.

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID. SELNOT
000003*
000004* PROGRAMA QUE EJECUTA QUERY PESADA
000005*
000006*
000007*
000008*-----------------------------------------------------------------
000009 ENVIRONMENT DIVISION.
000010 CONFIGURATION SECTION.
000011 SOURCE-COMPUTER.        MAINFR.
000012 OBJECT-COMPUTER.        MAINFR.
000013 SPECIAL-NAMES.
000014
000015         DECIMAL-POINT IS COMMA.
000016
000017 INPUT-OUTPUT SECTION.
000018 FILE-CONTROL.
000019
000020
000021
000022
000023 DATA DIVISION.
000024
000025 FILE SECTION.
000026
000027
000028
000029 WORKING-STORAGE SECTION.
000030
000031 01 COUNTR                  PIC X(10).
000032 01 RETCOUNT                PIC S9(10) USAGE COMP-3.
000033****************** VARIABLES AUXILIARES **************************
000034 01  WK-VARIABLES.
000035     02  WK-STATUS-FILEPRU          PIC X(2)       VALUE '00'.
000036     02  WK-RESTO                   PIC 9(10)v9(7) VALUE ZEROS.
000037     02  WK-NUM-OPER                PIC 9(10)      VALUE ZEROS.
000038     02  WK-BIG                     PIC S9(9)V9(7) VALUE ZEROS.
000039
000040     02  WK-NUM-FICH                PIC 9(10)      VALUE ZEROS.
000041     02  WK-NUM-PRUE                PIC 9(2)       VALUE ZEROS.
000042
000043****************** COPY DE LOG DE IMPRESION **********************
000044 01  WK-LOG.
000045     02  WK-MSGCAB1                 PIC X(80).
000046     02  WK-MSGCAB2                 PIC X(80).
000047     02  WK-MSGCAB3                 PIC X(80).
000048     02  WK-MSG1                    PIC X(23).
000049     02  WK-HORA-TIME               PIC X(2).
000050     02  WK-MSG2                    PIC X(10).
000051     02  WK-MIN-TIME                PIC X(2).
000052     02  WK-MSG3                    PIC X(11).
000053     02  WK-SEC-TIME                PIC X(2).
000054     02  WK-MSG4                    PIC X(12).
000055     02  WK-CEN-TIME                PIC X(2).
000056     02  WK-MSG5                    PIC X(6).
```

```
000057     02  WK-INSTR                    PIC X(30).
000058     02  WK-MSG6                     PIC X(80).
000059
000060***************** VARIABLES TEMPORALES *************************
000061 01  WK-VAR-TIEMPO.
000062     02  WK-MARCA-INI                PIC 9(9)V9(2)  VALUE ZEROS.
000063     02  WK-MARCA-FIN                PIC 9(9)V9(2)  VALUE ZEROS.
000064     02  WK-MARCA                    PIC 9(9)V9(2)  VALUE ZEROS.
000065     02  WK-MARCA-INIF               PIC 9(9)V9(2)  VALUE ZEROS.
000066     02  WK-MARCA-FINF               PIC 9(9)V9(2)  VALUE ZEROS.
000067     02  WK-CALC-SEGUNDOS            PIC 9(9)       VALUE ZEROS.
000068     02  WK-CALC-CENTESIMAS          PIC 9(9)V9(2)  VALUE ZEROS.
000069     02  WK-CALC-MINUTOS             PIC 9(9)       VALUE ZEROS.
000070     02  WK-CALC-HORA                PIC 9(9)       VALUE ZEROS.
000071     02  WK-INSTRUCTION              PIC X(30)      VALUE SPACE.
000072     02  WK-MARCA-TMP.
000073         03  WK-MARCA-HORA           PIC 9(2)       VALUE ZEROS.
000074         03  WK-MARCA-MIN            PIC 9(2)       VALUE ZEROS.
000075         03  WK-MARCA-SEC            PIC 9(2)       VALUE ZEROS.
000076         03  WK-MARCA-CEN            PIC 9(2)       VALUE ZEROS.
000077***************** CONSTANTES NUMERICAS *************************
000078 01  CN-CONSTANTES.
000079     02  CN-BIG                      PIC S9(9)V9(7)
000080                                           VALUE 987654321,8554469.
000081     02  CN-NUM-OPER                 PIC 9(10)      VALUE 1000000.
000082     02  CN-NUM-FICH                 PIC 9(10)      VALUE 60000000.
000083     02  CN-NUM-TEST                 PIC 9(02)      VALUE 8.
000084***************** CONSTANTES ALFANUMERICAS *********************
000085 01  CA-CONSTANTES.
000086     02  CONSTANTE                   PIC X(2)       VALUE 'AA'.
000087     02  STRING1                     PIC X(5)       VALUE 'PABLO'.
000088     02  STRING2                     PIC X(5)       VALUE 'PEREZ'.
000089     02  STRING-AUX                  PIC X(5)       VALUE SPACES.
000090
000091
000092
000093*------------------DB2 area-----------------------------------
000094      EXEC SQL
000095
000096          INCLUDE SQLCA
000097
000098      END-EXEC.
000099*
000100*****************************************************************
000101* DCLGEN TABLE(KC02520.EMP)                                      *
000102*        LIBRARY(KC02520.DCLEMP.COBOL)                            *
000103*        LANGUAGE(COBOL)                                          *
000104*        QUOTE                                                    *
000105* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS   *
000106*****************************************************************
000107     EXEC SQL DECLARE KC02520.EMP TABLE
000108     ( EMPNO                          CHAR(6) NOT NULL,
000109       FIRSTNME                       CHAR(12) NOT NULL,
000110       MIDINIT                        CHAR(1) NOT NULL,
000111       LASTNAME                       CHAR(15) NOT NULL,
000112       WORKDEPT                       CHAR(3),
000113       PHONENO                        CHAR(4),
000114       HIREDATE                       DATE,
000115       JOB                            CHAR(8),
000116       EDLEVEL                        SMALLINT,
000117       SEX                            CHAR(1),
```

```
000118       BIRTHDATE                      DATE,
000119       SALARY                         DECIMAL(9, 2),
000120       BONUS                          DECIMAL(9, 2),
000121       COMM                           DECIMAL(9, 2)
000122     ) END-EXEC.
000123******************************************************************
000124* COBOL DECLARATION FOR TABLE KC02520.EMP                        *
000125******************************************************************
000126 01  DCLEMP.
000127     10 EMPNON                PIC X(6).
000128     10 FIRSTNME              PIC X(12).
000129     10 MIDINIT              PIC X(1).
000130     10 LASTNAME              PIC X(15).
000131     10 WORKDEPT             PIC X(3).
000132     10 PHONENO              PIC X(4).
000133     10 HIREDATE             PIC X(10).
000134     10 JOB                  PIC X(8).
000135     10 EDLEVEL              PIC S9(4) USAGE COMP.
000136     10 SEX                  PIC X(1).
000137     10 BIRTHDATE            PIC X(10).
000138     10 SALARY               PIC S9(7)V9(2) USAGE COMP-3.
000139     10 BONUS                PIC S9(7)V9(2) USAGE COMP-3.
000140     10 COMM                 PIC S9(7)V9(2) USAGE COMP-3.
000141******************************************************************
000142* DCLGEN TABLE(KC02520.EMPPROJACT)                               *
000143*        LIBRARY(KC02520.DCLEMP.COBOL)                            *
000144*        ACTION(REPLACE)                                          *
000145*        LANGUAGE(COBOL)                                          *
000146*        QUOTE                                                    *
000147* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS   *
000148******************************************************************
000149     EXEC SQL DECLARE KC02520.EMPPROJACT TABLE
000150     ( EMPNO                          CHAR(6) NOT NULL,
000151       PROJNO                         CHAR(6) NOT NULL,
000152       ACTNO                          SMALLINT NOT NULL,
000153       EMPTIME                        DECIMAL(5, 2),
000154       EMSTDATE                       DATE,
000155       EMENDATE                       DATE
000156     ) END-EXEC.
000157******************************************************************
000158* COBOL DECLARATION FOR TABLE KC02520.EMPPROJACT                 *
000159******************************************************************
000160 01  DCLEMPPROJACT.
000161     10 EMPNO                PIC X(6).
000162     10 PROJNO               PIC X(6).
000163     10 ACTNO                PIC S9(4) USAGE COMP.
000164     10 EMPTIME              PIC S9(3)V9(2) USAGE COMP-3.
000165     10 EMSTDATE             PIC X(10).
000166     10 EMENDATE             PIC X(10).
000167******************************************************************
000168* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 6       *
000169******************************************************************
000170*----------------------------------------------------------------
000171 PROCEDURE DIVISION.
000172*----------------------------------------------------------------
000173*MAIN SECTION.
000174     PERFORM 1000-INICIO
000175        THRU 1000-INICIO-EXIT
000176*
000177     PERFORM 2000-PROCESO
000178        THRU 2000-PROCESO-EXIT
```

```
000179*
000180     PERFORM 3000-FIN
000181        THRU 3000-FIN-EXIT
000182*
000183     .
000184***************** DECLARACION DE PARRAFOS DEL PROGRAMA **********
000185*****************************************************************
000186 1000-INICIO.
000187*****************************************************************
000188* INICIALIZACION DE VARIABLES
000189*****************************************************************
000190     INITIALIZE WK-LOG
000191                WK-VARIABLES
000192                WK-VAR-TIEMPO
000193     INITIALIZE DCLEMP
000194     INITIALIZE COUNTR
000195                RETCOUNT
000196     DISPLAY '********************************* INICIO **********
000197-    '****************************'
000198     MOVE '*****************************************************
000199-    '**************************'          TO WK-MSGCAB1
000200     MOVE 'TIEMPO TRANSCURRIDO -->'        TO WK-MSG1
000201     MOVE ' MINUTOS: '                     TO WK-MSG2
000202     MOVE ' SEGUNDOS: '                    TO WK-MSG3
000203     MOVE ' CENTESIMAS '                   TO WK-MSG4
000204     MOVE ' PARA '                         TO WK-MSG5
000205*
000206     .
000207 1000-INICIO-EXIT.
000208     EXIT
000209*
000210     .
000211*****************************************************************
000212 2000-PROCESO.
000213*****************************************************************
000214* PROCESO DE PROGRAMA
000215*****************************************************************
000216     PERFORM GET-MARCA-TIEMP
000217        THRU GET-MARCA-TIEMP-EXIT
000218
000219     MOVE WK-MARCA                         TO WK-MARCA-INI
000220*
000221*
000222*
000223     EXEC SQL
000224
000225              SELECT COUNT(CE.EMPNO)
000226              INTO   :COUNTR, :RETCOUNT
000227              FROM (
000228                  SELECT B.EMPNO
000229                  FROM KC02520.EMP B
000230                  WHERE NOT EXISTS (
000231                      SELECT A.EMPNO
000232                      FROM KC02520.EMP A
000233                      WHERE NOT EXISTS (
000234                          SELECT EMPNO
000235                          FROM KC02520.EMPPROJACT T
000236                          )
000237                      )
000238                  ) CE
000239
```

```
000240
000241
000242     END-EXEC
000243*
000244
000245***** CALCULO DE DURACION DEL PROGRAMA
000246        PERFORM GET-MARCA-TIEMP
000247           THRU GET-MARCA-TIEMP-EXIT
000248
000249*
000250     .
000251 2000-PROCESO-EXIT.
000252     EXIT
000253*
000254     .
000255*
000256 GET-MARCA-TIEMP.
000257*
000258     ACCEPT WK-MARCA-TMP FROM TIME.
000259*
000260     COMPUTE WK-MARCA ROUNDED = ( WK-MARCA-HORA * 3600 )  +
000261     ( WK-MARCA-MIN * 60    )  +
000262     WK-MARCA-SEC              +
000263     ( WK-MARCA-CEN / 100   )
000264*
000265     .
000266 GET-MARCA-TIEMP-EXIT.
000267     EXIT.
000268*****************************************************************
000269 3000-FIN.
000270*****************************************************************
000271* FINAL DE PROGRAMA
000272*****************************************************************
000273****************** CALCULO DE DURACION DEL PROGRAMA **************
000274     MOVE 'DOBLE NOT EXISTS'               TO WK-INSTRUCTION
000275*****
000276     DISPLAY 'COUNT: ' COUNTR
000277     DISPLAY 'SQLCODE: ' SQLCODE
000278     PERFORM CALCULA-DURACION
000279        THRU CALCULA-DURACION-EXIT
000280*
000281     DISPLAY '********************************** FIN ************
000282-    '***************************'
000283     STOP RUN
000284*
000285     .
000286 3000-FIN-EXIT.
000287     EXIT
000288*
000289     .
000290*
000291 SET-MARCA-TIEMP.
000292*
000293     MOVE WK-MARCA                         TO WK-CALC-SEGUNDOS
000294     MOVE ZEROS                            TO WK-CALC-MINUTOS
000295*                                             WK-CALC-HORA
000296     DIVIDE WK-CALC-SEGUNDOS
000297     BY 60
000298     GIVING WK-CALC-MINUTOS
000299     REMAINDER WK-CALC-SEGUNDOS
000300*
```

```
000301      IF WK-CALC-MINUTOS > 60
000302         DIVIDE WK-CALC-MINUTOS BY 60
000303         GIVING WK-CALC-MINUTOS REMAINDER WK-CALC-HORA
000304      END-IF
000305*
000306      COMPUTE WK-CALC-CENTESIMAS = WK-CALC-SEGUNDOS - WK-MARCA
000307      COMPUTE WK-CALC-CENTESIMAS = WK-CALC-CENTESIMAS * 100
000308      COMPUTE WK-RESTO = WK-CALC-SEGUNDOS / 60
000309*
000310      MOVE WK-CALC-CENTESIMAS              TO WK-MARCA-CEN
000311      MOVE WK-CALC-HORA                    TO WK-MARCA-HORA
000312      MOVE WK-CALC-MINUTOS                 TO WK-MARCA-MIN
000313      MOVE WK-CALC-SEGUNDOS                TO WK-MARCA-SEC
000314*
000315      .
000316 SET-MARCA-TIEMP-EXIT.
000317      EXIT.
000318*
000319 CALCULA-DURACION.
000320*
000321      MOVE WK-MARCA                        TO WK-MARCA-FIN
000322*
000323      COMPUTE WK-MARCA = WK-MARCA-FIN - WK-MARCA-INI
000324*
000325      PERFORM SET-MARCA-TIEMP
000326         THRU SET-MARCA-TIEMP-EXIT
000327*
000328      MOVE WK-MARCA-MIN                    TO WK-MIN-TIME
000329      MOVE WK-MARCA-SEC                    TO WK-SEC-TIME
000330      MOVE WK-MARCA-CEN                    TO WK-CEN-TIME
000331      MOVE WK-INSTRUCTION                  TO WK-INSTR
000332      DISPLAY WK-LOG
000333*
000334      .
000335*
000336 CALCULA-DURACION-EXIT.
000337      EXIT.
000338*
000339
000340
000341
000342 END PROGRAM SELNOT.
```

Nombre del Programa: **SELCRUZ**
Descripción: Programa que cuantifica el tiempo que tarda la consulta con doble cruce.

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID. SELCRUZ
000003*
000004* PROGRAMA QUE EJECUTA QUERY PESADA
000005*
000006*
000007*
000008*----------------------------------------------------------------
000009 ENVIRONMENT DIVISION.
000010 CONFIGURATION SECTION.
000011 SOURCE-COMPUTER.        MAINFR.
000012 OBJECT-COMPUTER.        MAINFR.
000013 SPECIAL-NAMES.
```

```
000014
000015         DECIMAL-POINT IS COMMA.
000016
000017 INPUT-OUTPUT SECTION.
000018 FILE-CONTROL.
000019
000020
000021
000022
000023 DATA DIVISION.
000024
000025 FILE SECTION.
000026
000027
000028
000029 WORKING-STORAGE SECTION.
000030
000031 01 COUNTR                  PIC X(10).
000032 01 RETCOUNT                PIC S9(10) USAGE COMP-3.
000033****************** VARIABLES AUXILIARES **************************
000034 01  WK-VARIABLES.
000035     02  WK-STATUS-FILEPRU          PIC X(2)       VALUE '00'.
000036     02  WK-RESTO                   PIC 9(10)v9(7) VALUE ZEROS.
000037     02  WK-NUM-OPER                PIC 9(10)      VALUE ZEROS.
000038     02  WK-BIG                     PIC S9(9)V9(7) VALUE ZEROS.
000039
000040     02  WK-NUM-FICH                PIC 9(10)      VALUE ZEROS.
000041     02  WK-NUM-PRUE                PIC 9(2)       VALUE ZEROS.
000042
000043****************** COPY DE LOG DE IMPRESION **********************
000044 01  WK-LOG.
000045     02  WK-MSGCAB1                 PIC X(80).
000046     02  WK-MSGCAB2                 PIC X(80).
000047     02  WK-MSGCAB3                 PIC X(80).
000048     02  WK-MSG1                    PIC X(23).
000049     02  WK-HORA-TIME               PIC X(2).
000050     02  WK-MSG2                    PIC X(10).
000051     02  WK-MIN-TIME                PIC X(2).
000052     02  WK-MSG3                    PIC X(11).
000053     02  WK-SEC-TIME                PIC X(2).
000054     02  WK-MSG4                    PIC X(12).
000055     02  WK-CEN-TIME                PIC X(2).
000056     02  WK-MSG5                    PIC X(6).
000057     02  WK-INSTR                   PIC X(30).
000058     02  WK-MSG6                    PIC X(80).
000059
000060****************** VARIABLES TEMPORALES **************************
000061 01  WK-VAR-TIEMPO.
000062     02  WK-MARCA-INI               PIC 9(9)V9(2)  VALUE ZEROS.
000063     02  WK-MARCA-FIN               PIC 9(9)V9(2)  VALUE ZEROS.
000064     02  WK-MARCA                   PIC 9(9)V9(2)  VALUE ZEROS.
000065     02  WK-MARCA-INIF              PIC 9(9)V9(2)  VALUE ZEROS.
000066     02  WK-MARCA-FINF              PIC 9(9)V9(2)  VALUE ZEROS.
000067     02  WK-CALC-SEGUNDOS           PIC 9(9)       VALUE ZEROS.
000068     02  WK-CALC-CENTESIMAS         PIC 9(9)V9(2)  VALUE ZEROS.
000069     02  WK-CALC-MINUTOS            PIC 9(9)       VALUE ZEROS.
000070     02  WK-CALC-HORA               PIC 9(9)       VALUE ZEROS.
000071     02  WK-INSTRUCTION             PIC X(30)      VALUE SPACE.
000072     02  WK-MARCA-TMP.
000073         03  WK-MARCA-HORA          PIC 9(2)       VALUE ZEROS.
000074         03  WK-MARCA-MIN           PIC 9(2)       VALUE ZEROS.
```

```
000075          03  WK-MARCA-SEC           PIC 9(2)       VALUE ZEROS.
000076          03  WK-MARCA-CEN           PIC 9(2)       VALUE ZEROS.
000077****************** CONSTANTES NUMERICAS ************************
000078 01  CN-CONSTANTES.
000079     02  CN-BIG                      PIC S9(9)V9(7)
000080                                           VALUE 987654321,8554469.
000081     02  CN-NUM-OPER                 PIC 9(10)      VALUE 1000000.
000082     02  CN-NUM-FICH                 PIC 9(10)      VALUE 60000000.
000083     02  CN-NUM-TEST                 PIC 9(02)      VALUE 8.
000084****************** CONSTANTES ALFANUMERICAS ********************
000085 01  CA-CONSTANTES.
000086     02  CONSTANTE                   PIC X(2)       VALUE 'AA'.
000087     02  STRING1                     PIC X(5)       VALUE 'PABLO'.
000088     02  STRING2                     PIC X(5)       VALUE 'PEREZ'.
000089     02  STRING-AUX                  PIC X(5)       VALUE SPACES.
000090
000091
000092*------------------DB2 area-------------------------------------
000093      EXEC SQL
000094
000095          INCLUDE SQLCA
000096
000097      END-EXEC.
000098*
000099*****************************************************************
000100* DCLGEN TABLE(KC02520.EMP)                                      *
000101*        LIBRARY(KC02520.DCLEMP.COBOL)                           *
000102*        LANGUAGE(COBOL)                                         *
000103*        QUOTE                                                   *
000104* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS   *
000105*****************************************************************
000106     EXEC SQL DECLARE KC02520.EMP TABLE
000107     ( EMPNO                          CHAR(6) NOT NULL,
000108       FIRSTNME                       CHAR(12) NOT NULL,
000109       MIDINIT                        CHAR(1) NOT NULL,
000110       LASTNAME                       CHAR(15) NOT NULL,
000111       WORKDEPT                       CHAR(3),
000112       PHONENO                        CHAR(4),
000113       HIREDATE                       DATE,
000114       JOB                            CHAR(8),
000115       EDLEVEL                        SMALLINT,
000116       SEX                            CHAR(1),
000117       BIRTHDATE                      DATE,
000118       SALARY                         DECIMAL(9, 2),
000119       BONUS                          DECIMAL(9, 2),
000120       COMM                           DECIMAL(9, 2)
000121     ) END-EXEC.
000122*****************************************************************
000123* COBOL DECLARATION FOR TABLE KC02520.EMP                        *
000124*****************************************************************
000125 01  DCLEMP.
000126     10 EMPNON                PIC X(6).
000127     10 FIRSTNME              PIC X(12).
000128     10 MIDINIT              PIC X(1).
000129     10 LASTNAME              PIC X(15).
000130     10 WORKDEPT             PIC X(3).
000131     10 PHONENO              PIC X(4).
000132     10 HIREDATE             PIC X(10).
000133     10 JOB                  PIC X(8).
000134     10 EDLEVEL              PIC S9(4) USAGE COMP.
000135     10 SEX                  PIC X(1).
```

```
000136     10 BIRTHDATE            PIC X(10).
000137     10 SALARY               PIC S9(7)V9(2) USAGE COMP-3.
000138     10 BONUS                PIC S9(7)V9(2) USAGE COMP-3.
000139     10 COMM                 PIC S9(7)V9(2) USAGE COMP-3.
000140************************************************************
000141* DCLGEN TABLE(KC02520.PROJ)                                *
000142*        LIBRARY(KC02520.DCLPROJ.COBOL)                      *
000143*        LANGUAGE(COBOL)                                     *
000144*        QUOTE                                               *
000145* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS   *
000146************************************************************
000147     EXEC SQL DECLARE KC02520.PROJ TABLE
000148     ( PROJNO                        CHAR(6) NOT NULL,
000149       PROJNAME                      CHAR(24) NOT NULL,
000150       DEPTNO                        CHAR(3) NOT NULL,
000151       RESPEMP                       CHAR(6) NOT NULL,
000152       PRSTAFF                       DECIMAL(5, 2),
000153       PRSTDATE                      DATE,
000154       PRENDATE                      DATE,
000155       MAJPROJ                       CHAR(6)
000156     ) END-EXEC.
000157************************************************************
000158* COBOL DECLARATION FOR TABLE DSN81010.PROJ                 *
000159************************************************************
000160 01  DCLPROJ.
000161     10 PROJNON               PIC X(6).
000162     10 PROJNAME              PIC X(24).
000163     10 DEPTNO               PIC X(3).
000164     10 RESPEMP              PIC X(6).
000165     10 PRSTAFF              PIC S9(3)V9(2) USAGE COMP-3.
000166     10 PRSTDATE             PIC X(10).
000167     10 PRENDATE             PIC X(10).
000168     10 MAJPROJ              PIC X(6).
000169************************************************************
000170* DCLGEN TABLE(KC02520.EMPPROJACT)                          *
000171*        LIBRARY(KC02520.DCLEMP.COBOL)                       *
000172*        ACTION(REPLACE)                                     *
000173*        LANGUAGE(COBOL)                                     *
000174*        QUOTE                                               *
000175* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS   *
000176************************************************************
000177     EXEC SQL DECLARE KC02520.EMPPROJACT TABLE
000178     ( EMPNO                         CHAR(6) NOT NULL,
000179       PROJNO                        CHAR(6) NOT NULL,
000180       ACTNO                         SMALLINT NOT NULL,
000181       EMPTIME                       DECIMAL(5, 2),
000182       EMSTDATE                      DATE,
000183       EMENDATE                      DATE
000184     ) END-EXEC.
000185************************************************************
000186* COBOL DECLARATION FOR TABLE KC02520.EMPPROJACT            *
000187************************************************************
000188 01  DCLEMPPROJACT.
000189     10 EMPNO                PIC X(6).
000190     10 PROJNO               PIC X(6).
000191     10 ACTNO                PIC S9(4) USAGE COMP.
000192     10 EMPTIME              PIC S9(3)V9(2) USAGE COMP-3.
000193     10 EMSTDATE             PIC X(10).
000194     10 EMENDATE             PIC X(10).
000195************************************************************
000196* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 6  *
```

```
000197*****************************************************************
000198*----------------------------------------------------------------
000199 PROCEDURE DIVISION.
000200*----------------------------------------------------------------
000201*MAIN SECTION.
000202     PERFORM 1000-INICIO
000203        THRU 1000-INICIO-EXIT
000204*
000205     PERFORM 2000-PROCESO
000206        THRU 2000-PROCESO-EXIT
000207*
000208     PERFORM 3000-FIN
000209        THRU 3000-FIN-EXIT
000210*
000211     .
000212***************** DECLARACION DE PARRAFOS DEL PROGRAMA **********
000213*****************************************************************
000214 1000-INICIO.
000215*****************************************************************
000216* INICIALIZACION DE VARIABLES
000217*****************************************************************
000218     INITIALIZE WK-LOG
000219                WK-VARIABLES
000220                WK-VAR-TIEMPO
000221     INITIALIZE DCLEMP
000222     INITIALIZE COUNTR
000223                RETCOUNT
000224     DISPLAY '********************************** INICIO **********
000225-    '***************************'
000226     MOVE '*****************************************************
000227-    '*************************'          TO WK-MSGCAB1
000228     MOVE 'TIEMPO TRANSCURRIDO -->'       TO WK-MSG1
000229     MOVE ' MINUTOS: '                    TO WK-MSG2
000230     MOVE ' SEGUNDOS: '                   TO WK-MSG3
000231     MOVE ' CENTESIMAS '                  TO WK-MSG4
000232     MOVE ' PARA '                        TO WK-MSG5
000233*
000234     .
000235 1000-INICIO-EXIT.
000236     EXIT
000237*
000238     .
000239*****************************************************************
000240 2000-PROCESO.
000241*****************************************************************
000242* PROCESO DE PROGRAMA
000243*****************************************************************
000244     PERFORM GET-MARCA-TIEMP
000245        THRU GET-MARCA-TIEMP-EXIT
000246
000247     MOVE WK-MARCA                        TO WK-MARCA-INI
000248*
000249*
000250*
000251*
000252*
000253     EXEC SQL
000254
000255
000256*
000257             SELECT COUNT(T.EMPNO)
```

```
000258              INTO   :COUNTR, :RETCOUNT
000259              FROM (
000260                  SELECT R.EMPNO
000261                  FROM (
000262                      SELECT P.PROJNO, P.PROJNAME, E.EMPNO
000263                      FROM KC02520.PROJ P
000264                      INNER JOIN
000265                      KC02520.EMPPROJACT E
000266                      ON P.PROJNO = E.PROJNO
000267                      ) R
000268                  INNER JOIN KC02520.EMP EM
000269                  ON R.EMPNO = EM.EMPNO
000270                  ) T
000271
000272
000273     END-EXEC
000274*
000275
000276***** CALCULO DE DURACION DEL PROGRAMA
000277        PERFORM GET-MARCA-TIEMP
000278           THRU GET-MARCA-TIEMP-EXIT
000279
000280*
000281     .
000282 2000-PROCESO-EXIT.
000283     EXIT
000284*
000285     .
000286*
000287 GET-MARCA-TIEMP.
000288*
000289     ACCEPT WK-MARCA-TMP FROM TIME.
000290*
000291     COMPUTE WK-MARCA ROUNDED = ( WK-MARCA-HORA * 3600 )  +
000292     ( WK-MARCA-MIN * 60    )  +
000293     WK-MARCA-SEC              +
000294     ( WK-MARCA-CEN / 100   )
000295*
000296     .
000297 GET-MARCA-TIEMP-EXIT.
000298     EXIT.
000299*****************************************************************
000300 3000-FIN.
000301*****************************************************************
000302* FINAL DE PROGRAMA
000303*****************************************************************
000304****************** CALCULO DE DURACION DEL PROGRAMA **************
000305     MOVE 'DOBLE CRUCE DE TABLAS'         TO WK-INSTRUCTION
000306*****
000307     DISPLAY 'COUNT: ' COUNTR
000308     DISPLAY 'SQLCODE: ' SQLCODE
000309     DISPLAY 'RETCOUNT: ' RETCOUNT
000310     PERFORM CALCULA-DURACION
000311        THRU CALCULA-DURACION-EXIT
000312*
000313     DISPLAY '********************************** FIN ************
000314-    '***************************'
000315     STOP RUN
000316*
000317     .
000318 3000-FIN-EXIT.
```

```
000319     EXIT
000320*
000321     .
000322*
000323 SET-MARCA-TIEMP.
000324*
000325     MOVE WK-MARCA                          TO WK-CALC-SEGUNDOS
000326     MOVE ZEROS                             TO WK-CALC-MINUTOS
000327*                                               WK-CALC-HORA
000328     DIVIDE WK-CALC-SEGUNDOS
000329     BY 60
000330     GIVING WK-CALC-MINUTOS
000331     REMAINDER WK-CALC-SEGUNDOS
000332*
000333     IF WK-CALC-MINUTOS > 60
000334        DIVIDE WK-CALC-MINUTOS BY 60
000335        GIVING WK-CALC-MINUTOS REMAINDER WK-CALC-HORA
000336     END-IF
000337*
000338     COMPUTE WK-CALC-CENTESIMAS = WK-CALC-SEGUNDOS - WK-MARCA
000339     COMPUTE WK-CALC-CENTESIMAS = WK-CALC-CENTESIMAS * 100
000340     COMPUTE WK-RESTO = WK-CALC-SEGUNDOS / 60
000341*
000342     MOVE WK-CALC-CENTESIMAS                TO WK-MARCA-CEN
000343     MOVE WK-CALC-HORA                      TO WK-MARCA-HORA
000344     MOVE WK-CALC-MINUTOS                   TO WK-MARCA-MIN
000345     MOVE WK-CALC-SEGUNDOS                  TO WK-MARCA-SEC
000346*
000347     .
000348 SET-MARCA-TIEMP-EXIT.
000349     EXIT.
000350*
000351 CALCULA-DURACION.
000352*
000353     MOVE WK-MARCA                          TO WK-MARCA-FIN
000354*
000355     COMPUTE WK-MARCA = WK-MARCA-FIN - WK-MARCA-INI
000356*
000357     PERFORM SET-MARCA-TIEMP
000358        THRU SET-MARCA-TIEMP-EXIT
000359*
000360     MOVE WK-MARCA-MIN                      TO WK-MIN-TIME
000361     MOVE WK-MARCA-SEC                      TO WK-SEC-TIME
000362     MOVE WK-MARCA-CEN                      TO WK-CEN-TIME
000363     MOVE WK-INSTRUCTION                    TO WK-INSTR
000364     DISPLAY WK-LOG
000365*
000366     .
000367*
000368 CALCULA-DURACION-EXIT.
000369     EXIT.
000370*
000371
000372 END PROGRAM SELCRUZ.
```

Nombre del Programa: **SELAVG**
Descripción: Programa que cuantifica el tiempo que tarda la consulta con funciones agregadas.

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID. SELAVG
000003*
000004* PROGRAMA QUE EJECUTA QUERY PESADA
000005*
000006*
000007*
000008*----------------------------------------------------------------
000009 ENVIRONMENT DIVISION.
000010 CONFIGURATION SECTION.
000011 SOURCE-COMPUTER.        MAINFR.
000012 OBJECT-COMPUTER.        MAINFR.
000013 SPECIAL-NAMES.
000014
000015         DECIMAL-POINT IS COMMA.
000016
000017 INPUT-OUTPUT SECTION.
000018 FILE-CONTROL.
000019
000020
000021
000022
000023 DATA DIVISION.
000024
000025 FILE SECTION.
000026
000027
000028
000029 WORKING-STORAGE SECTION.
000030
000031 01 COUNTR                          PIC X(10).
000032 01 AVGRES                          PIC X(10).
000033 01 RETCOUNT                        PIC S9(10) USAGE COMP-3.
000034 01 AVGR                            PIC S9(10) USAGE COMP-3.
000035 01 RETAVG                          PIC S9(10) USAGE COMP-3.
000036***************** VARIABLES AUXILIARES ************************
000037 01  WK-VARIABLES.
000038     02  WK-STATUS-FILEPRU          PIC X(2)       VALUE '00'.
000039     02  WK-RESTO                   PIC 9(10)v9(7) VALUE ZEROS.
000040     02  WK-NUM-OPER                PIC 9(10)      VALUE ZEROS.
000041     02  WK-BIG                     PIC S9(9)V9(7) VALUE ZEROS.
000042
000043     02  WK-NUM-FICH                PIC 9(10)      VALUE ZEROS.
000044     02  WK-NUM-PRUE                PIC 9(2)       VALUE ZEROS.
000045
000046***************** COPY DE LOG DE IMPRESION *********************
000047 01  WK-LOG.
000048     02  WK-MSGCAB1                 PIC X(80).
000049     02  WK-MSGCAB2                 PIC X(80).
000050     02  WK-MSGCAB3                 PIC X(80).
000051     02  WK-MSG1                    PIC X(23).
000052     02  WK-HORA-TIME               PIC X(2).
000053     02  WK-MSG2                    PIC X(10).
000054     02  WK-MIN-TIME                PIC X(2).
000055     02  WK-MSG3                    PIC X(11).
```

```
000056      02  WK-SEC-TIME                PIC X(2).
000057      02  WK-MSG4                    PIC X(12).
000058      02  WK-CEN-TIME                PIC X(2).
000059      02  WK-MSG5                    PIC X(6).
000060      02  WK-INSTR                   PIC X(30).
000061      02  WK-MSG6                    PIC X(80).
000062
000063****************** VARIABLES TEMPORALES **************************
000064 01  WK-VAR-TIEMPO.
000065      02  WK-MARCA-INI               PIC 9(9)V9(2)  VALUE ZEROS.
000066      02  WK-MARCA-FIN               PIC 9(9)V9(2)  VALUE ZEROS.
000067      02  WK-MARCA                   PIC 9(9)V9(2)  VALUE ZEROS.
000068      02  WK-MARCA-INIF              PIC 9(9)V9(2)  VALUE ZEROS.
000069      02  WK-MARCA-FINF              PIC 9(9)V9(2)  VALUE ZEROS.
000070      02  WK-CALC-SEGUNDOS           PIC 9(9)       VALUE ZEROS.
000071      02  WK-CALC-CENTESIMAS         PIC 9(9)V9(2)  VALUE ZEROS.
000072      02  WK-CALC-MINUTOS            PIC 9(9)       VALUE ZEROS.
000073      02  WK-CALC-HORA               PIC 9(9)       VALUE ZEROS.
000074      02  WK-INSTRUCTION             PIC X(30)      VALUE SPACE.
000075      02  WK-MARCA-TMP.
000076          03  WK-MARCA-HORA          PIC 9(2)       VALUE ZEROS.
000077          03  WK-MARCA-MIN           PIC 9(2)       VALUE ZEROS.
000078          03  WK-MARCA-SEC           PIC 9(2)       VALUE ZEROS.
000079          03  WK-MARCA-CEN           PIC 9(2)       VALUE ZEROS.
000080****************** CONSTANTES NUMERICAS **************************
000081 01  CN-CONSTANTES.
000082      02  CN-BIG                     PIC S9(9)V9(7)
000083                                           VALUE 987654321,8554469.
000084      02  CN-NUM-OPER                PIC 9(10)      VALUE 1000000.
000085      02  CN-NUM-FICH                PIC 9(10)      VALUE 60000000.
000086      02  CN-NUM-TEST                PIC 9(02)      VALUE 8.
000087****************** CONSTANTES ALFANUMERICAS **********************
000088 01  CA-CONSTANTES.
000089      02  CONSTANTE                  PIC X(2)       VALUE 'AA'.
000090      02  STRING1                    PIC X(5)       VALUE 'PABLO'.
000091      02  STRING2                    PIC X(5)       VALUE 'PEREZ'.
000092      02  STRING-AUX                 PIC X(5)       VALUE SPACES.
000093
000094
000095*------------------DB2 area-------------------------------------
000096       EXEC SQL
000097
000098           INCLUDE SQLCA
000099
000100       END-EXEC.
000101*
000102*****************************************************************
000103* DCLGEN TABLE(KC02520.EMP)                                      *
000104*        LIBRARY(KC02520.DCLEMP.COBOL)                           *
000105*        LANGUAGE(COBOL)                                         *
000106*        QUOTE                                                   *
000107* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS   *
000108*****************************************************************
000109      EXEC SQL DECLARE KC02520.EMP TABLE
000110      ( EMPNO                           CHAR(6) NOT NULL,
000111        FIRSTNME                        CHAR(12) NOT NULL,
000112        MIDINIT                         CHAR(1) NOT NULL,
000113        LASTNAME                        CHAR(15) NOT NULL,
000114        WORKDEPT                        CHAR(3),
000115        PHONENO                         CHAR(4),
000116        HIREDATE                        DATE,
```

```
000117       JOB                            CHAR(8),
000118       EDLEVEL                        SMALLINT,
000119       SEX                            CHAR(1),
000120       BIRTHDATE                      DATE,
000121       SALARY                         DECIMAL(9, 2),
000122       BONUS                          DECIMAL(9, 2),
000123       COMM                           DECIMAL(9, 2)
000124     ) END-EXEC.
000125******************************************************************
000126* COBOL DECLARATION FOR TABLE KC02520.EMP                        *
000127******************************************************************
000128 01  DCLEMP.
000129     10 EMPNON                PIC X(6).
000130     10 FIRSTNME              PIC X(12).
000131     10 MIDINIT              PIC X(1).
000132     10 LASTNAME              PIC X(15).
000133     10 WORKDEPT             PIC X(3).
000134     10 PHONENO              PIC X(4).
000135     10 HIREDATE             PIC X(10).
000136     10 JOB                  PIC X(8).
000137     10 EDLEVEL              PIC S9(4) USAGE COMP.
000138     10 SEX                  PIC X(1).
000139     10 BIRTHDATE            PIC X(10).
000140     10 SALARY               PIC S9(7)V9(2) USAGE COMP-3.
000141     10 BONUS                PIC S9(7)V9(2) USAGE COMP-3.
000142     10 COMM                 PIC S9(7)V9(2) USAGE COMP-3.
000143******************************************************************
000144* DCLGEN TABLE(KC02520.EMPPROJACT)                               *
000145*        LIBRARY(KC02520.DCLEMP.COBOL)                            *
000146*        ACTION(REPLACE)                                          *
000147*        LANGUAGE(COBOL)                                          *
000148*        QUOTE                                                    *
000149* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS   *
000150******************************************************************
000151     EXEC SQL DECLARE KC02520.EMPPROJACT TABLE
000152     ( EMPNO                          CHAR(6) NOT NULL,
000153       PROJNO                         CHAR(6) NOT NULL,
000154       ACTNO                          SMALLINT NOT NULL,
000155       EMPTIME                        DECIMAL(5, 2),
000156       EMSTDATE                       DATE,
000157       EMENDATE                       DATE
000158     ) END-EXEC.
000159******************************************************************
000160* COBOL DECLARATION FOR TABLE KC02520.EMPPROJACT                 *
000161******************************************************************
000162 01  DCLEMPPROJACT.
000163     10 EMPNO                PIC X(6).
000164     10 PROJNO               PIC X(6).
000165     10 ACTNO                PIC S9(4) USAGE COMP.
000166     10 EMPTIME              PIC S9(3)V9(2) USAGE COMP-3.
000167     10 EMSTDATE             PIC X(10).
000168     10 EMENDATE             PIC X(10).
000169******************************************************************
000170* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 6       *
000171******************************************************************
000172*----------------------------------------------------------------
000173 PROCEDURE DIVISION.
000174*----------------------------------------------------------------
000175*MAIN SECTION.
000176     PERFORM 1000-INICIO
000177        THRU 1000-INICIO-EXIT
```

```
000178*
000179      PERFORM 2000-PROCESO
000180         THRU 2000-PROCESO-EXIT
000181*
000182      PERFORM 3000-FIN
000183         THRU 3000-FIN-EXIT
000184*
000185      .
000186***************** DECLARACION DE PARRAFOS DEL PROGRAMA **********
000187*****************************************************************
000188 1000-INICIO.
000189*****************************************************************
000190* INICIALIZACION DE VARIABLES
000191*****************************************************************
000192      INITIALIZE WK-LOG
000193                 WK-VARIABLES
000194                 WK-VAR-TIEMPO
000195      INITIALIZE DCLEMP
000196      INITIALIZE COUNTR
000197                 RETCOUNT
000198      DISPLAY '********************************** INICIO **********
000199-     '**************************'
000200      MOVE '*************************************************
000201-     '*************************'          TO WK-MSGCAB1
000202      MOVE 'TIEMPO TRANSCURRIDO -->'       TO WK-MSG1
000203      MOVE ' MINUTOS: '                    TO WK-MSG2
000204      MOVE ' SEGUNDOS: '                   TO WK-MSG3
000205      MOVE ' CENTESIMAS '                  TO WK-MSG4
000206      MOVE ' PARA '                        TO WK-MSG5
000207*
000208      .
000209 1000-INICIO-EXIT.
000210      EXIT
000211*
000212      .
000213*****************************************************************
000214 2000-PROCESO.
000215*****************************************************************
000216* PROCESO DE PROGRAMA
000217*****************************************************************
000218      PERFORM GET-MARCA-TIEMP
000219         THRU GET-MARCA-TIEMP-EXIT
000220
000221      MOVE WK-MARCA                        TO WK-MARCA-INI
000222*
000223*
000224*
000225*
000226      EXEC SQL
000227
000228
000229
000230               SELECT COUNT(AV.EMPNO)
000231               INTO   :COUNTR, :RETCOUNT
000232               FROM (
000233                   SELECT EMPNO, AVG(EMPTIME)
000234                   FROM KC02520.EMPPROJACT
000235                   GROUP BY EMPNO
000236                   HAVING AVG(EMPTIME) < 5
000237                  ) AV
000238
```

```
000239
000240     END-EXEC
000241*
000242
000243***** CALCULO DE DURACION DEL PROGRAMA
000244        PERFORM GET-MARCA-TIEMP
000245           THRU GET-MARCA-TIEMP-EXIT
000246
000247*
000248     .
000249 2000-PROCESO-EXIT.
000250     EXIT
000251*
000252     .
000253*
000254 GET-MARCA-TIEMP.
000255*
000256     ACCEPT WK-MARCA-TMP FROM TIME.
000257*
000258     COMPUTE WK-MARCA ROUNDED = ( WK-MARCA-HORA * 3600 )  +
000259     ( WK-MARCA-MIN * 60    )  +
000260     WK-MARCA-SEC              +
000261     ( WK-MARCA-CEN / 100   )
000262*
000263     .
000264 GET-MARCA-TIEMP-EXIT.
000265     EXIT.
000266*****************************************************************
000267 3000-FIN.
000268*****************************************************************
000269* FINAL DE PROGRAMA
000270*****************************************************************
000271****************** CALCULO DE DURACION DEL PROGRAMA **************
000272     MOVE 'FUNC AGREGADAS '               TO WK-INSTRUCTION
000273*****
000274     DISPLAY 'CONTADOR DE MEDIAS (AVG): ' COUNTR
000275     DISPLAY 'SQLCODE: ' SQLCODE
000276     DISPLAY 'RETAVG: ' RETCOUNT
000277     PERFORM CALCULA-DURACION
000278        THRU CALCULA-DURACION-EXIT
000279*
000280     DISPLAY '********************************** FIN ************
000281-    '***************************'
000282     STOP RUN
000283*
000284     .
000285 3000-FIN-EXIT.
000286     EXIT
000287*
000288     .
000289*
000290 SET-MARCA-TIEMP.
000291*
000292     MOVE WK-MARCA                        TO WK-CALC-SEGUNDOS
000293     MOVE ZEROS                           TO WK-CALC-MINUTOS
000294*                                             WK-CALC-HORA
000295     DIVIDE WK-CALC-SEGUNDOS
000296     BY 60
000297     GIVING WK-CALC-MINUTOS
000298     REMAINDER WK-CALC-SEGUNDOS
000299*
```

```
000300     IF WK-CALC-MINUTOS > 60
000301        DIVIDE WK-CALC-MINUTOS BY 60
000302        GIVING WK-CALC-MINUTOS REMAINDER WK-CALC-HORA
000303     END-IF
000304*
000305     COMPUTE WK-CALC-CENTESIMAS = WK-CALC-SEGUNDOS - WK-MARCA
000306     COMPUTE WK-CALC-CENTESIMAS = WK-CALC-CENTESIMAS * 100
000307     COMPUTE WK-RESTO = WK-CALC-SEGUNDOS / 60
000308*
000309     MOVE WK-CALC-CENTESIMAS              TO WK-MARCA-CEN
000310     MOVE WK-CALC-HORA                    TO WK-MARCA-HORA
000311     MOVE WK-CALC-MINUTOS                 TO WK-MARCA-MIN
000312     MOVE WK-CALC-SEGUNDOS                TO WK-MARCA-SEC
000313*
000314     .
000315 SET-MARCA-TIEMP-EXIT.
000316     EXIT.
000317*
000318 CALCULA-DURACION.
000319*
000320     MOVE WK-MARCA                        TO WK-MARCA-FIN
000321*
000322     COMPUTE WK-MARCA = WK-MARCA-FIN - WK-MARCA-INI
000323*
000324     PERFORM SET-MARCA-TIEMP
000325        THRU SET-MARCA-TIEMP-EXIT
000326*
000327     MOVE WK-MARCA-MIN                    TO WK-MIN-TIME
000328     MOVE WK-MARCA-SEC                    TO WK-SEC-TIME
000329     MOVE WK-MARCA-CEN                    TO WK-CEN-TIME
000330     MOVE WK-INSTRUCTION                  TO WK-INSTR
000331     DISPLAY WK-LOG
000332*
000333     .
000334*
000335 CALCULA-DURACION-EXIT.
000336     EXIT.
000337*
000338
000339
000340 END PROGRAM SELAVG.
```

Nombre del Programa: **INSPROJ**
Descripción: Programa que inserta N registros a la tabla PROYECTO.

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID. INSPROJ
000003*
000004* PROGRAMA QUE INSETA EN TABLA PROYECTO
000005*
000006*
000007*
000008*----------------------------------------------------------------
000009 ENVIRONMENT DIVISION.
000010 CONFIGURATION SECTION.
000011 SOURCE-COMPUTER.        MAINFR.
000012 OBJECT-COMPUTER.        MAINFR.
000013 SPECIAL-NAMES.
000014
000015         DECIMAL-POINT IS COMMA.
```

```
000016
000017 INPUT-OUTPUT SECTION.
000018 FILE-CONTROL.
000019
000020
000021
000022
000023 DATA DIVISION.
000024
000025 FILE SECTION.
000026
000027
000028
000029 WORKING-STORAGE SECTION.
000030 01 STATUS-BENCH              PIC XX     VALUE '00'.
000031 01 RESTO                    PIC 9(10)v9(7) VALUE ZEROS.
000032 01 XTIMES                   PIC 9(10)  VALUE 1000000.
000033 01 PROJNO-AUX               PIC 9(6)   VALUE 0.
000034
000035
000036 01  WK-LOG.
000037    02 WK-MSG1               PIC X(13).
000038    02 WK-HORA-TIME          PIC X(2).
000039    02 WK-MSG2               PIC X(5).
000040    02 WK-MIN-TIME           PIC X(2).
000041    02 WK-MSG3               PIC X(6).
000042    02 WK-SEC-TIME           PIC X(2).
000043    02 WK-MSG4               PIC X(12).
000044    02 WK-CEN-TIME           PIC X(2).
000045    02 WK-MSG5               PIC X(5).
000046    02 WK-INSTR              PIC X(30).
000047
000048
000049 01 STRING1                  PIC X(5) VALUE 'PABLO'.
000050 01 STRING2                  PIC X(5) VALUE 'PEREZ'.
000051 01 STRING_AUX               PIC X(5) VALUE SPACES.
000052 01 CA-SELECT                PIC X(6) VALUE 'SELECT'.
000053*-------------------DB2 area-------------------------------------
000054      EXEC SQL
000055
000056          INCLUDE SQLCA
000057
000058      END-EXEC.
000059*
000060*****************************************************************
000061* DCLGEN TABLE(KC02520.PROJ)                                     *
000062*        LIBRARY(KC02520.DCLPROJ.COBOL)                           *
000063*        LANGUAGE(COBOL)                                          *
000064*        QUOTE                                                    *
000065* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS   *
000066*****************************************************************
000067     EXEC SQL DECLARE KC02520.PROJ TABLE
000068     ( PROJNO                        CHAR(6) NOT NULL,
000069       PROJNAME                      CHAR(24) NOT NULL,
000070       DEPTNO                        CHAR(3) NOT NULL,
000071       RESPEMP                       CHAR(6) NOT NULL,
000072       PRSTAFF                       DECIMAL(5, 2),
000073       PRSTDATE                      DATE,
000074       PRENDATE                      DATE,
000075       MAJPROJ                       CHAR(6)
000076     ) END-EXEC.
```

```
000077*****************************************************************
000078* COBOL DECLARATION FOR TABLE KC02520.PROJ                       *
000079*****************************************************************
000080 01  DCLPROJ.
000081     10 PROJNO              PIC X(6).
000082     10 PROJNAME             PIC X(24).
000083     10 DEPTNO               PIC X(3).
000084     10 RESPEMP              PIC X(6).
000085     10 PRSTAFF              PIC S9(3)V9(2) USAGE COMP-3.
000086     10 PRSTDATE             PIC X(10).
000087     10 PRENDATE             PIC X(10).
000088     10 MAJPROJ              PIC X(6).
000089 LINKAGE SECTION.
000090 01  DFHCOMMAREA                PIC X.
000091*---------------------------------------------------------------
000092 PROCEDURE DIVISION.
000093*---------------------------------------------------------------
000094*MAIN SECTION.
000095     PERFORM 1000-INICIO
000096        THRU 1000-INICIO-EXIT
000097*
000098     PERFORM 2000-PROCESO
000099        THRU 2000-PROCESO-EXIT
000100*
000101     PERFORM 3000-FIN
000102        THRU 3000-FIN-EXIT
000103*
000104     .
000105***************** DECLARACION DE PARRAFOS DEL PROGRAMA **********
000106*****************************************************************
000107 1000-INICIO.
000108*****************************************************************
000109* INICIALIZACION DE VARIABLES
000110*****************************************************************
000111
000112     DISPLAY '********************************* INICIO **********
000113-    '***************************'
000114     INITIALIZE DCLPROJ
000115     MOVE +1 TO PROJNO-AUX
000116     MOVE PROJNO-AUX TO PROJNO
000117     MOVE 'PROYECTO PRUEBA' TO PROJNAME
000118     MOVE '001' TO DEPTNO
000119     MOVE '001KCT' TO RESPEMP
000120     MOVE +500 TO PRSTAFF
000121     MOVE '2015-03-03' TO PRSTDATE
000122     MOVE '2012-07-27' TO PRENDATE
000123     MOVE '001001' TO MAJPROJ
000124
000125*
000126     .
000127 1000-INICIO-EXIT.
000128     EXIT
000129*
000130     .
000131*****************************************************************
000132 2000-PROCESO.
000133*****************************************************************
000134* PROCESO DE PROGRAMA
000135*****************************************************************
000136**
000137     PERFORM 900000 TIMES
```

```
000138     EXEC SQL
000139          INSERT INTO KC02520.PROJ
000140                   (PROJNO,
000141                    PROJNAME,
000142                    DEPTNO,
000143                    RESPEMP,
000144                    PRSTAFF,
000145                    PRSTDATE,
000146                    PRENDATE,
000147                    MAJPROJ)
000148             VALUES(:PROJNO,
000149                    :PROJNAME,
000150                    :DEPTNO,
000151                    :RESPEMP,
000152                    :PRSTAFF,
000153                    :PRSTDATE,
000154                    :PRENDATE,
000155                    :MAJPROJ)
000156     END-EXEC
000157
000158     IF SQLCODE NOT= ZERO
000159          DISPLAY 'REGISTRO ERRONEO: '
000160          DISPLAY SQLCODE
000161          DISPLAY DCLPROJ
000162     END-IF
000163     ADD 1 TO PROJNO-AUX
000164     MOVE PROJNO-AUX TO PROJNO
000165     END-PERFORM
000166
000167*
000168     .
000169 2000-PROCESO-EXIT.
000170     EXIT
000171*
000172     .
000173*
000174*****************************************************************
000175 3000-FIN.
000176*****************************************************************
000177* FINAL DE PROGRAMA
000178*****************************************************************
000179
000180     DISPLAY 'ULTIMO REGISTRO INTRODUCIDO '
000181     DISPLAY 'DCLG: ' DCLPROJ
000182     DISPLAY 'SQLCODE: ' SQLCODE
000183     DISPLAY '********************************* FIN ************
000184-    '***************************'
000185     STOP RUN
000186*
000187     .
000188 3000-FIN-EXIT.
000189     EXIT
000190*
000191     .
000192
000193
000194 END PROGRAM INSPROJ.
```

Nombre del Programa: **INSEMP**
Descripción: Programa que inserta N registros a la tabla EMPLEADO.

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID. INSEMP
000003*
000004* PROGRAMA QUE INSETA EN TABLA EMPLEADO
000005*
000006*
000007*
000008*-----------------------------------------------------------------
000009 ENVIRONMENT DIVISION.
000010 CONFIGURATION SECTION.
000011 SOURCE-COMPUTER.        MAINFR.
000012 OBJECT-COMPUTER.        MAINFR.
000013 SPECIAL-NAMES.
000014
000015         DECIMAL-POINT IS COMMA.
000016
000017 INPUT-OUTPUT SECTION.
000018 FILE-CONTROL.
000019
000020
000021
000022
000023 DATA DIVISION.
000024
000025 FILE SECTION.
000026
000027
000028
000029 WORKING-STORAGE SECTION.
000030 01 STATUS-BENCH              PIC XX     VALUE '00'.
000031 01 RESTO                     PIC 9(10)v9(7) VALUE ZEROS.
000032 01 XTIMES                    PIC 9(10)  VALUE 1000000.
000033 01 EMPNO-AUX                 PIC 9(6)   VALUE 0.
000034
000035
000036 01  WK-LOG.
000037    02 WK-MSG1                PIC X(13).
000038    02 WK-HORA-TIME           PIC X(2).
000039    02 WK-MSG2                PIC X(5).
000040    02 WK-MIN-TIME            PIC X(2).
000041    02 WK-MSG3                PIC X(6).
000042    02 WK-SEC-TIME            PIC X(2).
000043    02 WK-MSG4                PIC X(12).
000044    02 WK-CEN-TIME            PIC X(2).
000045    02 WK-MSG5                PIC X(5).
000046    02 WK-INSTR               PIC X(30).
000047
000048
000049 01 STRING1                   PIC X(5) VALUE 'PABLO'.
000050 01 STRING2                   PIC X(5) VALUE 'PEREZ'.
000051 01 STRING_AUX                PIC X(5) VALUE SPACES.
000052 01 CA-SELECT                 PIC X(6) VALUE 'SELECT'.
000053*------------------DB2 area-------------------------------------
000054      EXEC SQL
000055
000056          INCLUDE SQLCA
000057
```

```
000058      END-EXEC.
000059*
000060******************************************************************
000061* DCLGEN TABLE(KC02520.EMP)                                      *
000062*        LIBRARY(KC02520.DCLEMP.COBOL)                            *
000063*        LANGUAGE(COBOL)                                          *
000064*        QUOTE                                                    *
000065* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS   *
000066******************************************************************
000067     EXEC SQL DECLARE KC02520.EMP TABLE
000068     ( EMPNO                          CHAR(6) NOT NULL,
000069       FIRSTNME                       CHAR(12) NOT NULL,
000070       MIDINIT                        CHAR(1) NOT NULL,
000071       LASTNAME                       CHAR(15) NOT NULL,
000072       WORKDEPT                       CHAR(3),
000073       PHONENO                        CHAR(4),
000074       HIREDATE                       DATE,
000075       JOB                            CHAR(8),
000076       EDLEVEL                        SMALLINT,
000077       SEX                            CHAR(1),
000078       BIRTHDATE                      DATE,
000079       SALARY                         DECIMAL(9, 2),
000080       BONUS                          DECIMAL(9, 2),
000081       COMM                           DECIMAL(9, 2)
000082     ) END-EXEC.
000083******************************************************************
000084* COBOL DECLARATION FOR TABLE KC02520.EMP                        *
000085******************************************************************
000086 01  DCLEMP.
000087     10 EMPNO               PIC X(6).
000088     10 FIRSTNME            PIC X(12).
000089     10 MIDINIT             PIC X(1).
000090     10 LASTNAME            PIC X(15).
000091     10 WORKDEPT            PIC X(3).
000092     10 PHONENO             PIC X(4).
000093     10 HIREDATE            PIC X(10).
000094     10 JOB                 PIC X(8).
000095     10 EDLEVEL             PIC S9(4) USAGE COMP.
000096     10 SEX                 PIC X(1).
000097     10 BIRTHDATE           PIC X(10).
000098     10 SALARY              PIC S9(7)V9(2) USAGE COMP-3.
000099     10 BONUS               PIC S9(7)V9(2) USAGE COMP-3.
000100     10 COMM                PIC S9(7)V9(2) USAGE COMP-3.
000101
000102 LINKAGE SECTION.
000103 01  DFHCOMMAREA                PIC X.
000104*-----------------------------------------------------------------
000105 PROCEDURE DIVISION.
000106*-----------------------------------------------------------------
000107*MAIN SECTION.
000108     PERFORM 1000-INICIO
000109        THRU 1000-INICIO-EXIT
000110*
000111     PERFORM 2000-PROCESO
000112        THRU 2000-PROCESO-EXIT
000113*
000114     PERFORM 3000-FIN
000115        THRU 3000-FIN-EXIT
000116*
000117     .
000118***************** DECLARACION DE PARRAFOS DEL PROGRAMA **********
```

```
000119*****************************************************************
000120 1000-INICIO.
000121*****************************************************************
000122* INICIALIZACION DE VARIABLES
000123*****************************************************************
000124
000125     INITIALIZE WK-LOG
000126     INITIALIZE DCLEMP
000127*
000128     DISPLAY '********************************* INICIO **********
000129-    '***************************'
000130     MOVE +1 TO EMPNO-AUX
000131     MOVE EMPNO-AUX TO EMPNO
000132     MOVE '0' TO MIDINIT
000133     MOVE 'PABLO' TO FIRSTNME
000134     MOVE 'PEREZ' TO LASTNAME
000135     MOVE '001' TO WORKDEPT
000136     MOVE '1013' TO PHONENO
000137     MOVE '2015-03-03' TO HIREDATE
000138     MOVE 'STUDENT ' TO JOB
000139     MOVE +1 TO EDLEVEL
000140     MOVE 'M' TO SEX
000141     MOVE '1990-07-27' TO BIRTHDATE
000142     MOVE +500 TO SALARY
000143     MOVE +500 TO BONUS
000144     MOVE +500 TO COMM
000145*
000146     .
000147 1000-INICIO-EXIT.
000148     EXIT
000149*
000150     .
000151*****************************************************************
000152 2000-PROCESO.
000153*****************************************************************
000154* PROCESO DE PROGRAMA
000155*****************************************************************
000156**
000157     PERFORM 900000 TIMES
000158     EXEC SQL
000159          INSERT INTO KC02520.EMP
000160                  (EMPNO,
000161                   FIRSTNME,
000162                   MIDINIT,
000163                   LASTNAME,
000164                   WORKDEPT,
000165                   PHONENO,
000166                   HIREDATE,
000167                   JOB,
000168                   EDLEVEL,
000169                   SEX,
000170                   BIRTHDATE,
000171                   SALARY,
000172                   BONUS,
000173                   COMM)
000174             VALUES(:EMPNO,
000175                    :FIRSTNME,
000176                    :MIDINIT,
000177                    :LASTNAME,
000178                    :WORKDEPT,
000179                    :PHONENO,
```

```
000180                    :HIREDATE,
000181                    :JOB,
000182                    :EDLEVEL,
000183                    :SEX,
000184                    :BIRTHDATE,
000185                    :SALARY,
000186                    :BONUS,
000187                    :COMM)
000188     END-EXEC
000189
000190     IF SQLCODE NOT= ZERO
000191          DISPLAY 'REGISTRO ERRONEO: '
000192          DISPLAY SQLCODE
000193          DISPLAY DCLEMP
000194     END-IF
000195     ADD 1 TO EMPNO-AUX
000196     MOVE EMPNO-AUX TO EMPNO
000197     END-PERFORM
000198
000199*
000200     .
000201 2000-PROCESO-EXIT.
000202     EXIT
000203*
000204     .
000205*
000206*****************************************************************
000207 3000-FIN.
000208*****************************************************************
000209* FINAL DE PROGRAMA
000210*****************************************************************
000211     DISPLAY 'ULTIMO REGISTRO INTRODUCIDO '
000212     DISPLAY 'DCLG: ' DCLEMP
000213     DISPLAY 'SQLCODE: ' SQLCODE
000214     DISPLAY '********************************** FIN ************
000215-    '***************************'
000216     STOP RUN
000217*
000218     .
000219 3000-FIN-EXIT.
000220     EXIT
000221*
000222     .
000207
000208 END PROGRAM INSEMP.
```

Nombre del Programa: **INSEMAC**
Descripción: Programa que inserta N registros a la tabla EMPPROJACT.

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID. INSEMAC
000003*
000004* PROGRAMA QUE INSETA EN TABLA EMPLEADO-PROYECTO-ACT
000005*
000006*
000007*
000008*----------------------------------------------------------------
000009 ENVIRONMENT DIVISION.
000010 CONFIGURATION SECTION.
000011 SOURCE-COMPUTER.        MAINFR.
```

```
000012 OBJECT-COMPUTER.        MAINFR.
000013 SPECIAL-NAMES.
000014
000015         DECIMAL-POINT IS COMMA.
000016
000017 INPUT-OUTPUT SECTION.
000018 FILE-CONTROL.
000019
000020
000021
000022
000023 DATA DIVISION.
000024
000025 FILE SECTION.
000026
000027
000028
000029 WORKING-STORAGE SECTION.
000030 01 STATUS-BENCH              PIC XX     VALUE '00'.
000031 01 RESTO                     PIC 9(10)v9(7) VALUE ZEROS.
000032 01 XTIMES                    PIC 9(10)  VALUE 1000000.
000033 01 PROJNO-AUX                PIC 9(6)   VALUE 0.
000034 01 EMPNO-AUX                 PIC 9(6)   VALUE 0.
000035 01 ACTNO-AUX                 PIC 9(6)   VALUE 0.
000036
000037
000038 01  WK-LOG.
000039    02 WK-MSG1                PIC X(13).
000040    02 WK-HORA-TIME           PIC X(2).
000041    02 WK-MSG2                PIC X(5).
000042    02 WK-MIN-TIME            PIC X(2).
000043    02 WK-MSG3                PIC X(6).
000044    02 WK-SEC-TIME            PIC X(2).
000045    02 WK-MSG4                PIC X(12).
000046    02 WK-CEN-TIME            PIC X(2).
000047    02 WK-MSG5                PIC X(5).
000048    02 WK-INSTR               PIC X(30).
000049
000050
000051 01 STRING1                   PIC X(5) VALUE 'PABLO'.
000052 01 STRING2                   PIC X(5) VALUE 'PEREZ'.
000053 01 STRING_AUX                PIC X(5) VALUE SPACES.
000054 01 CA-SELECT                 PIC X(6) VALUE 'SELECT'.
000055*------------------DB2 area-------------------------------------
000056      EXEC SQL
000057
000058          INCLUDE SQLCA
000059
000060      END-EXEC.
000061*
000062******************************************************************
000063* DCLGEN TABLE(KC02520.EMPPROJACT)                               *
000064*        LIBRARY(KC02520.DCLEMP.COBOL)                            *
000065*        ACTION(REPLACE)                                          *
000066*        LANGUAGE(COBOL)                                          *
000067*        QUOTE                                                    *
000068* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS   *
000069******************************************************************
000070     EXEC SQL DECLARE KC02520.EMPPROJACT TABLE
000071     ( EMPNO                          CHAR(6) NOT NULL,
000072       PROJNO                         CHAR(6) NOT NULL,
```

```
000073       ACTNO                          SMALLINT NOT NULL,
000074       EMPTIME                        DECIMAL(5, 2),
000075       EMSTDATE                       DATE,
000076       EMENDATE                       DATE
000077     ) END-EXEC.
000078******************************************************************
000079* COBOL DECLARATION FOR TABLE KC02520.EMPPROJACT                  *
000080******************************************************************
000081 01  DCLEMPPROJACT.
000082     10 EMPNO                PIC X(6).
000083     10 PROJNO               PIC X(6).
000084     10 ACTNO                PIC S9(4) USAGE COMP.
000085     10 EMPTIME              PIC S9(3)V9(2) USAGE COMP-3.
000086     10 EMSTDATE             PIC X(10).
000087     10 EMENDATE             PIC X(10).
000088******************************************************************
000089 LINKAGE SECTION.
000090 01  DFHCOMMAREA                PIC X.
000091*----------------------------------------------------------------
000092 PROCEDURE DIVISION.
000093*----------------------------------------------------------------
000094*MAIN SECTION.
000095     PERFORM 1000-INICIO
000096        THRU 1000-INICIO-EXIT
000097*
000098     PERFORM 2000-PROCESO
000099        THRU 2000-PROCESO-EXIT
000100*
000101     PERFORM 3000-FIN
000102        THRU 3000-FIN-EXIT
000103*
000104     .
000105***************** DECLARACION DE PARRAFOS DEL PROGRAMA **********
000106******************************************************************
000107 1000-INICIO.
000108******************************************************************
000109* INICIALIZACION DE VARIABLES
000110******************************************************************
000111
000112
000113     DISPLAY '********************************* INICIO **********
000114-    '***************************'
000115     INITIALIZE DCLEMPPROJACT
000116     MOVE +1 TO EMPNO-AUX
000117     MOVE EMPNO-AUX TO EMPNO
000118     MOVE +1 TO PROJNO-AUX
000119     MOVE PROJNO-AUX TO PROJNO
000120     MOVE +1 TO ACTNO
000121     MOVE +3 TO EMPTIME
000122     MOVE '2015-03-03' TO EMSTDATE
000123     MOVE '2012-07-27' TO EMENDATE
000124
000125
000126*
000127     .
000128 1000-INICIO-EXIT.
000129     EXIT
000130*
000131     .
000132******************************************************************
000133 2000-PROCESO.
```

```
000134*****************************************************************
000135* PROCESO DE PROGRAMA
000136*****************************************************************
000137**
000138     PERFORM 900000 TIMES
000139     EXEC SQL
000140          INSERT INTO KC02520.EMPPROJACT
000141                   (EMPNO,
000142                    PROJNO,
000143                    ACTNO,
000144                    EMPTIME,
000145                    EMSTDATE,
000146                    EMENDATE)
000147             VALUES(:EMPNO,
000148                    :PROJNO,
000149                    :ACTNO,
000150                    :EMPTIME,
000151                    :EMSTDATE,
000152                    :EMENDATE)
000153     END-EXEC
000154
000155     IF SQLCODE NOT= ZERO
000156          DISPLAY 'REGISTRO ERRONEO: '
000157          DISPLAY SQLCODE
000158          DISPLAY DCLEMPPROJACT
000159     END-IF
000160     ADD 1 TO PROJNO-AUX
000161     MOVE PROJNO-AUX TO PROJNO
000162     ADD 1 TO ACTNO
000163     END-PERFORM
000164
000165
000166
000167*
000168     .
000169 2000-PROCESO-EXIT.
000170     EXIT
000171*
000172     .
000173*
000174*****************************************************************
000175 3000-FIN.
000176*****************************************************************
000177* FINAL DE PROGRAMA
000178*****************************************************************
000179
000180     DISPLAY 'ULTIMO REGISTRO INTRODUCIDO '
000181     DISPLAY 'DCLG: ' DCLEMPPROJACT
000182     DISPLAY 'SQLCODE: ' SQLCODE
000183     DISPLAY '********************************** FIN ************
000184-    '***************************'
000185     STOP RUN
000186*
000187     .
000188 3000-FIN-EXIT.
000189     EXIT
000190*
000191     .
000192
000193
000194 END PROGRAM INSEMAC.
```

M Scripts y consultas SQL

A continuación se detallan las tablas SQL.

- Declaración de la tabla EMPLEADO: En esta tabla se encuentran los datos de los empleados.

```
CREATE TABLE EMP
   ( EMPNO                CHAR(6) NOT NULL,
     FIRSTNME             CHAR(12) NOT NULL,
     MIDINIT              CHAR(1) NOT NULL,
     LASTNAME             CHAR(15) NOT NULL,
     WORKDEPT             CHAR(3),
     PHONENO              CHAR(4),
     HIREDATE             DATE,
     JOB                  CHAR(8),
     EDLEVEL              SMALLINT,
     SEX                  CHAR(1),
     BIRTHDATE            DATE,
     SALARY               DECIMAL(9, 2),
     BONUS                DECIMAL(9, 2),
     COMM                 DECIMAL(9, 2),
     PRIMARY KEY(EMPNO));
```

- Declaración de la tabla PROYECTO: En esta tabla se encuentran los datos de los proyectos.

```
CREATE TABLE PROJ
   ( PROJNO               CHAR(6) NOT NULL,
     PROJNAME             CHAR(24) NOT NULL,
     DEPTNO               CHAR(3) NOT NULL,
     RESPEMP              CHAR(6) NOT NULL,
     PRSTAFF              DECIMAL(5, 2),
     PRSTDATE             DATE,
     PRENDATE             DATE,
     MAJPROJ              CHAR(6),
     PRIMARY KEY(PROJNO));
```

- Declaración de la tabla EMPPROJACT: Esta tabla es la que relaciona a los empleados con los proyectos.

```
CREATE TABLE EMPPROJACT
   ( EMPNO                CHAR(6) NOT NULL,
     PROJNO               CHAR(6) NOT NULL,
     ACTNO                SMALLINT NOT NULL,
     EMPTIME              DECIMAL(5, 2),
     EMSTDATE             DATE,
     EMENDATE             DATE,
     PRIMARY KEY (EMPNO, PROJNO, ACTNO),
```

```
FOREIGN KEY (EMPNO)
REFERENCES KC02520.EMP
ON DELETE  RESTRICT,
FOREIGN KEY (PROJNO)
REFERENCES KC02520.PROJ
ON DELETE CASCADE);
```

N Salida de los programas de prueba COBOL & C

```
Número de segundos transcurridos en la operacion  ESCRITURA: 12.297100 s
Número de segundos transcurridos en la operacion  LECTURA: 4.555500 s
Número de segundos transcurridos en la operacion  DIVISION: 0.030500 s
Número de segundos transcurridos en la operacion  SUMA: 0.022000 s
Número de segundos transcurridos en la operacion  RESTA: 0.019100 s
Número de segundos transcurridos en la operacion  MULTIPLICACION: 0.023400 s
Número de segundos transcurridos en la operacion  POTENCIA: 0.307100 s
Número de segundos transcurridos en la operacion  SWAP: 0.027900 s
Número de segundos transcurridos desde el comienzo del programa: 22.000000 s
```

Figura 0-11 Resultados obtenidos por el programa de medición de operaciones C (BATEC) en RDZ

```
*******************************************************************************
TIEMPO TRANSCURRIDO -->   MINUTOS: 00 SEGUNDOS: 00 CENTESIMAS 08 PARA COMPUTE-SUM
*******************************************************************************
TIEMPO TRANSCURRIDO -->   MINUTOS: 00 SEGUNDOS: 00 CENTESIMAS 07 PARA COMPUTE-RES
*******************************************************************************
TIEMPO TRANSCURRIDO -->   MINUTOS: 00 SEGUNDOS: 00 CENTESIMAS 14 PARA COMPUTE-DIV
*******************************************************************************
TIEMPO TRANSCURRIDO -->   MINUTOS: 00 SEGUNDOS: 00 CENTESIMAS 09 PARA COMPUTE-MUL
*******************************************************************************
TIEMPO TRANSCURRIDO -->   MINUTOS: 00 SEGUNDOS: 01 CENTESIMAS 87 PARA COMPUTE-POT
*******************************************************************************
TIEMPO TRANSCURRIDO -->   MINUTOS: 00 SEGUNDOS: 00 CENTESIMAS 01 PARA SWAP
*******************************************************************************
TIEMPO TRANSCURRIDO -->   MINUTOS: 00 SEGUNDOS: 02 CENTESIMAS 93 PARA ESCRIBIR
*******************************************************************************
TIEMPO TRANSCURRIDO -->   MINUTOS: 00 SEGUNDOS: 02 CENTESIMAS 91 PARA LEER
*******************************************************************************
TIEMPO TRANSCURRIDO -->   MINUTOS: 00 SEGUNDOS: 08 CENTESIMAS 12 PARA TEST COMPLETADO
********************************** FIN ************************************
```

Figura 0-12 Resultados obtenidos por el programa de medición de operaciones COBOL (BATECOB) en RDZ

O Salida de los programas de prueba COBOL–DB2

```
********************************** INICIO ***************************************
CONTADOR DE MEDIAS (AVG): 1
SQLCODE: 0000000000
RETAVG: 0000000000
*****************************************************************************

TIEMPO TRANSCURRIDO -->   MINUTOS: 00 SEGUNDOS: 00 CENTESIMAS 04 PARA FUNC AGREGADAS

********************************** FIN ****************************************
```

Figura 0-13 Resultados obtenidos por el programa de medición de SQL COBOL (SELAVG) en RDZ para tamaño de tabla de 10.000 registros

```
********************************** INICIO ***************************************
CONTADOR DE MEDIAS (AVG): 1
SQLCODE: 0000000000
RETAVG: 0000000000
*****************************************************************************

TIEMPO TRANSCURRIDO -->   MINUTOS: 00 SEGUNDOS: 00 CENTESIMAS 21 PARA FUNC AGREGADAS

********************************** FIN ****************************************
```

Figura 0-14 Resultados obtenidos por el programa de medición de SQL COBOL (SELAVG) en RDZ para tamaño de tabla de 100.000 registros

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+-
********************************** INICIO ***************************************
CONTADOR DE MEDIAS (AVG): 1
SQLCODE: 0000000000
RETAVG: 0000000000
*****************************************************************************

TIEMPO TRANSCURRIDO -->   MINUTOS: 00 SEGUNDOS: 01 CENTESIMAS 06 PARA FUNC AGREGADAS

********************************** FIN ****************************************
```

Figura 0-15 Resultados obtenidos por el programa de medición de SQL COBOL (SELAVG) en RDZ para tamaño de tabla de 900.000 registros

```
********************************** INICIO ****************************************
COUNT: 10000
SQLCODE: 0000000000
******************************************************************************

TIEMPO TRANSCURRIDO -->   MINUTOS: 00 SEGUNDOS: 00 CENTESIMAS 06 PARA DOBLE NOT EXISTS

*********************************** FIN ******************************************
```

Figura 0-16 Resultados obtenidos por el programa de medición de SQL COBOL (SELNOT) en RDZ para tamaño de tabla de 10.000 registros

```
********************************** INICIO ****************************************
COUNT: 100000
SQLCODE: 0000000000
******************************************************************************

TIEMPO TRANSCURRIDO -->   MINUTOS: 00 SEGUNDOS: 00 CENTESIMAS 59 PARA DOBLE NOT EXISTS

*********************************** FIN ******************************************
```

Figura 0-17 Resultados obtenidos por el programa de medición de SQL COBOL (SELNOT) en RDZ para tamaño de tabla de 100.000 registros

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+--
********************************** INICIO ****************************************
COUNT: 900000
SQLCODE: 0000000000
******************************************************************************

TIEMPO TRANSCURRIDO -->   MINUTOS: 00 SEGUNDOS: 04 CENTESIMAS 52 PARA DOBLE NOT EXISTS

*********************************** FIN ******************************************
```

Figura 0-18 Resultados obtenidos por el programa de medición de SQL COBOL (SELNOT) en RDZ para tamaño de tabla de 900.000 registros

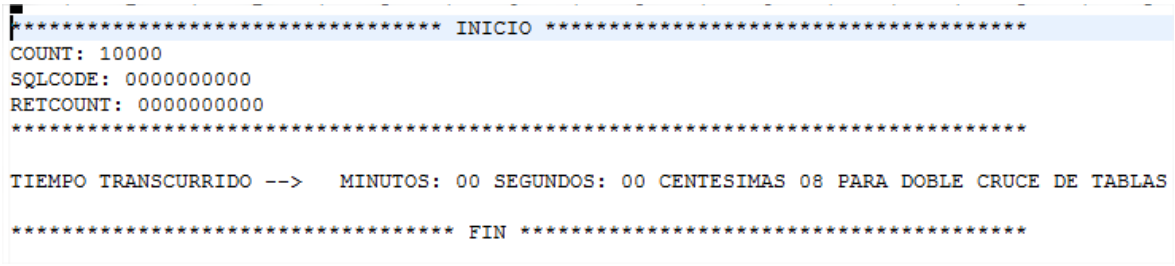


Figura 0-19 Resultados obtenidos por el programa de medición de SQL COBOL (SELCRUZ) en RDZ para tamaño de tabla de 10.000 registros

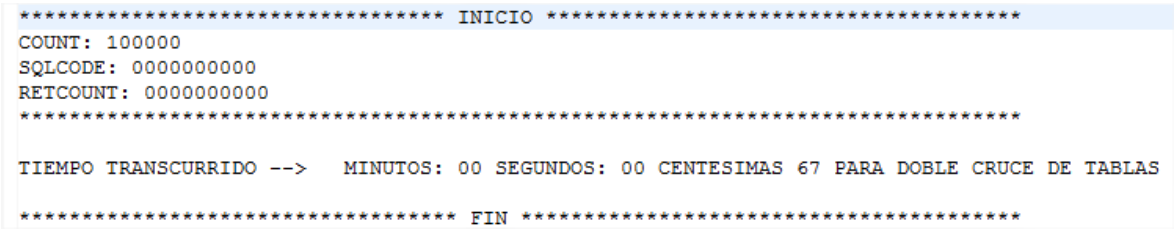


Figura 0-20 Resultados obtenidos por el programa de medición de SQL COBOL (SELCRUZ) en RDZ para tamaño de tabla de 100.000 registros

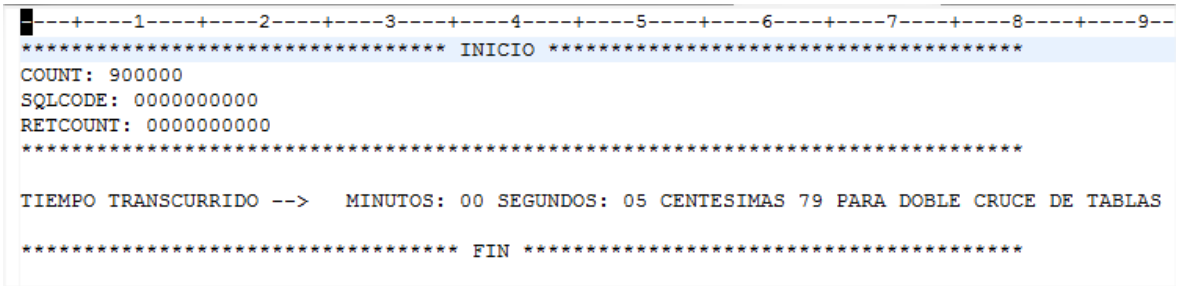


Figura 0-21 Resultados obtenidos por el programa de medición de SQL COBOL (SELCRUZ) en RDZ para tamaño de tabla de 900.000 registros